

Arm[®] Architecture Registers

Armv9, for Armv9-A architecture profile

arm

Arm® Architecture Registers

Armv9, for Armv9-A architecture profile

Copyright © 2018-2021 Arm Limited (or its affiliates). All rights reserved.

Release Information

For information on the change history and known issues for this release, see the Release notes in the System Register XML for Armv9 (2021-09)

Proprietary Notice

This document is protected by copyright and other related rights and the practice or implementation of the information contained in this document may be protected by one or more patents or pending patent applications. No part of this document may be reproduced in any form by any means without the express prior written permission of Arm. **No license, express or implied, by estoppel or otherwise to any intellectual property rights is granted by this document unless specifically stated.**

Your access to the information in this document is conditional upon your acceptance that you will not use or permit others to use the information for the purposes of determining whether implementations infringe any third party patents.

THIS DOCUMENT IS PROVIDED “AS IS”. ARM PROVIDES NO REPRESENTATIONS AND NO WARRANTIES, EXPRESS, IMPLIED OR STATUTORY, INCLUDING, WITHOUT LIMITATION, THE IMPLIED WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NON-INFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE WITH RESPECT TO THE DOCUMENT. For the avoidance of doubt, Arm makes no representation with respect to, and has undertaken no analysis to identify or understand the scope and content of, patents, copyrights, trade secrets, or other rights.

This document may include technical inaccuracies or typographical errors.

TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL ARM BE LIABLE FOR ANY DAMAGES, INCLUDING WITHOUT LIMITATION ANY DIRECT, INDIRECT, SPECIAL, INCIDENTAL, PUNITIVE, OR CONSEQUENTIAL DAMAGES, HOWEVER CAUSED AND REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF ANY USE OF THIS DOCUMENT, EVEN IF ARM HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

This document consists solely of commercial items. You shall be responsible for ensuring that any use, duplication or disclosure of this document complies fully with any relevant export laws and regulations to assure that this document or any portion thereof is not exported, directly or indirectly, in violation of such export laws. Use of the word “partner” in reference to Arm’s customers is not intended to create or refer to any partnership relationship with any other company. Arm may make changes to this document at any time and without notice.

This document may be translated into other languages for convenience, and you agree that if there is any conflict between the English version of this document and any translation, the terms of the English version of the Agreement shall prevail.

The Arm corporate logo and words marked with ® or ™ are registered trademarks or trademarks of Arm Limited (or its affiliates) in the US and/or elsewhere. All rights reserved. Other brands and names mentioned in this document may be the trademarks of their respective owners. You must follow the Arm’s trademark usage guidelines <http://www.arm.com/company/policies/trademarks>.

Copyright © 2018-2021 Arm Limited (or its affiliates). All rights reserved.
Arm Limited. Company 02557590 registered in England.
110 Fulbourn Road, Cambridge, England CB1 9NJ.

LES-PRE-20349 version 21.0

Confidentiality Status

This document is Non-Confidential. The right to use, copy and disclose this document may be subject to license restrictions in accordance with the terms of the agreement entered into by Arm and the party that Arm delivered this document to.

Product Status

This release covers multiple versions of the architecture. The content relating to different versions is given different quality ratings.

The information related to the 2021 Architecture Extensions and features FEAT_RME and FEAT_SME is at Alpha quality. Alpha quality means that most major features of the specification are described in the manual, some features and details might be missing.

The information related to the remaining Armv8-A features and the Armv9-A features (FEAT_SVE2, FEAT_TME, FEAT_TRBE, FEAT_BRBE and FEAT_ETE), which was also published in previous releases, is at Beta quality. Beta quality means that all major features of the specification are described, some details might be missing.

Web Address

<http://www.arm.com>

Progressive Terminology Commitment

Arm values inclusive communities. Arm recognizes that we and our industry have used terms that can be offensive. Arm strives to lead the industry and create change.

Previous issues of this document included terms that can be offensive. We have replaced these terms.

If you find offensive terms in this document, please contact terms@arm.com.

AArch64 System Registers

[ACCDATA_EL1](#): Accelerator Data

[ACTLR_EL1](#): Auxiliary Control Register (EL1)

[ACTLR_EL2](#): Auxiliary Control Register (EL2)

[ACTLR_EL3](#): Auxiliary Control Register (EL3)

[AFSR0_EL1](#): Auxiliary Fault Status Register 0 (EL1)

[AFSR0_EL2](#): Auxiliary Fault Status Register 0 (EL2)

[AFSR0_EL3](#): Auxiliary Fault Status Register 0 (EL3)

[AFSR1_EL1](#): Auxiliary Fault Status Register 1 (EL1)

[AFSR1_EL2](#): Auxiliary Fault Status Register 1 (EL2)

[AFSR1_EL3](#): Auxiliary Fault Status Register 1 (EL3)

[AIDR_EL1](#): Auxiliary ID Register

[ALLINT](#): All Interrupt Mask Bit

[AMAIR_EL1](#): Auxiliary Memory Attribute Indirection Register (EL1)

[AMAIR_EL2](#): Auxiliary Memory Attribute Indirection Register (EL2)

[AMAIR_EL3](#): Auxiliary Memory Attribute Indirection Register (EL3)

[AMCFGR_EL0](#): Activity Monitors Configuration Register

[AMCG1IDR_EL0](#): Activity Monitors Counter Group 1 Identification Register

[AMCGCR_EL0](#): Activity Monitors Counter Group Configuration Register

[AMCNTENCLR0_EL0](#): Activity Monitors Count Enable Clear Register 0

[AMCNTENCLR1_EL0](#): Activity Monitors Count Enable Clear Register 1

[AMCNTENSET0_EL0](#): Activity Monitors Count Enable Set Register 0

[AMCNTENSET1_EL0](#): Activity Monitors Count Enable Set Register 1

[AMCR_EL0](#): Activity Monitors Control Register

[AMEVCNTR0<n>_EL0](#): Activity Monitors Event Counter Registers 0

[AMEVCNTR1<n>_EL0](#): Activity Monitors Event Counter Registers 1

[AMEVCNTVOFF0<n>_EL2](#): Activity Monitors Event Counter Virtual Offset Registers 0

[AMEVCNTVOFF1<n>_EL2](#): Activity Monitors Event Counter Virtual Offset Registers 1

[AMEVTYPER0<n>_EL0](#): Activity Monitors Event Type Registers 0

[AMEVTYPER1<n>_EL0](#): Activity Monitors Event Type Registers 1

[AMUSERENR_EL0](#): Activity Monitors User Enable Register

[APDAKeyHi_EL1](#): Pointer Authentication Key A for Data (bits[127:64])

[APDAKeyLo_EL1](#): Pointer Authentication Key A for Data (bits[63:0])

[APDBKeyHi_EL1](#): Pointer Authentication Key B for Data (bits[127:64])

[APDBKeyLo_EL1](#): Pointer Authentication Key B for Data (bits[63:0])

[APGAKeyHi_EL1](#): Pointer Authentication Key A for Code (bits[127:64])

[APGAKeyLo_EL1](#): Pointer Authentication Key A for Code (bits[63:0])

[APIAKeyHi_EL1](#): Pointer Authentication Key A for Instruction (bits[127:64])

[APIAKeyLo_EL1](#): Pointer Authentication Key A for Instruction (bits[63:0])

[APIBKeyHi_EL1](#): Pointer Authentication Key B for Instruction (bits[127:64])

[APIBKeyLo_EL1](#): Pointer Authentication Key B for Instruction (bits[63:0])

[BRBCR_EL1](#): Branch Record Buffer Control Register (EL1)

[BRBCR_EL2](#): Branch Record Buffer Control Register (EL2)

[BRBFCR_EL1](#): Branch Record Buffer Function Control Register

[BRBIDR0_EL1](#): Branch Record Buffer ID0 Register

[BRBINF<n>_EL1](#): Branch Record Buffer Information Register <n>

[BRBINFINJ_EL1](#): Branch Record Buffer Information Injection Register

[BRBSRC<n>_EL1](#): Branch Record Buffer Source Address Register <n>

[BRBSRCINJ_EL1](#): Branch Record Buffer Source Address Injection Register

[BRBTGT<n>_EL1](#): Branch Record Buffer Target Address Register <n>

[BRBTGTINJ_EL1](#): Branch Record Buffer Target Address Injection Register

[BRBTS_EL1](#): Branch Record Buffer Timestamp Register

[CCSIDR2_EL1](#): Current Cache Size ID Register 2

[CCSIDR_EL1](#): Current Cache Size ID Register

[CLIDR_EL1](#): Cache Level ID Register

[CNTFRQ_EL0](#): Counter-timer Frequency register

[CNTHCTL_EL2](#): Counter-timer Hypervisor Control register

[CNTHPS_CTL_EL2](#): Counter-timer Secure Physical Timer Control register (EL2)

[CNTHPS_CVAL_EL2](#): Counter-timer Secure Physical Timer CompareValue register (EL2)

[CNTHPS_TVAL_EL2](#): Counter-timer Secure Physical Timer TimerValue register (EL2)

[CNTHP_CTL_EL2](#): Counter-timer Hypervisor Physical Timer Control register

[CNTHP_CVAL_EL2](#): Counter-timer Physical Timer CompareValue register (EL2)

[CNTHP_TVAL_EL2](#): Counter-timer Physical Timer TimerValue register (EL2)

[CNTHVS_CTL_EL2](#): Counter-timer Secure Virtual Timer Control register (EL2)

[CNTHVS_CVAL_EL2](#): Counter-timer Secure Virtual Timer CompareValue register (EL2)

[CNTHVS_TVAL_EL2](#): Counter-timer Secure Virtual Timer TimerValue register (EL2)

[CNTHV_CTL_EL2](#): Counter-timer Virtual Timer Control register (EL2)

[CNTHV_CVAL_EL2](#): Counter-timer Virtual Timer CompareValue register (EL2)

[CNTHV_TVAL_EL2](#): Counter-timer Virtual Timer TimerValue Register (EL2)

[CNTKCTL_EL1](#): Counter-timer Kernel Control register

[CNTPCTSS_EL0](#): Counter-timer Self-Synchronized Physical Count register

[CNTPCT_EL0](#): Counter-timer Physical Count register
[CNTPOFF_EL2](#): Counter-timer Physical Offset register
[CNTPS_CTL_EL1](#): Counter-timer Physical Secure Timer Control register
[CNTPS_CVAL_EL1](#): Counter-timer Physical Secure Timer CompareValue register
[CNTPS_TVAL_EL1](#): Counter-timer Physical Secure Timer TimerValue register
[CNTP_CTL_EL0](#): Counter-timer Physical Timer Control register
[CNTP_CVAL_EL0](#): Counter-timer Physical Timer CompareValue register
[CNTP_TVAL_EL0](#): Counter-timer Physical Timer TimerValue register
[CNTVCTSS_EL0](#): Counter-timer Self-Synchronized Virtual Count register
[CNTVCT_EL0](#): Counter-timer Virtual Count register
[CNTVOFF_EL2](#): Counter-timer Virtual Offset register
[CNTV_CTL_EL0](#): Counter-timer Virtual Timer Control register
[CNTV_CVAL_EL0](#): Counter-timer Virtual Timer CompareValue register
[CNTV_TVAL_EL0](#): Counter-timer Virtual Timer TimerValue register
[CONTEXTIDR_EL1](#): Context ID Register (EL1)
[CONTEXTIDR_EL2](#): Context ID Register (EL2)
[CPACR_EL1](#): Architectural Feature Access Control Register
[CPTR_EL2](#): Architectural Feature Trap Register (EL2)
[CPTR_EL3](#): Architectural Feature Trap Register (EL3)
[CSSELR_EL1](#): Cache Size Selection Register
[CTR_EL0](#): Cache Type Register
[CurrentEL](#): Current Exception Level
[DACR32_EL2](#): Domain Access Control Register
[DAIF](#): Interrupt Mask Bits
[DBGAUTHSTATUS_EL1](#): Debug Authentication Status register
[DBGBCR<n>_EL1](#): Debug Breakpoint Control Registers
[DBGBVR<n>_EL1](#): Debug Breakpoint Value Registers
[DBGCLAIMCLR_EL1](#): Debug CLAIM Tag Clear register
[DBGCLAIMSET_EL1](#): Debug CLAIM Tag Set register
[DBGDTRRX_EL0](#): Debug Data Transfer Register, Receive
[DBGDTRTX_EL0](#): Debug Data Transfer Register, Transmit
[DBGDTR_EL0](#): Debug Data Transfer Register, half-duplex
[DBGPRCR_EL1](#): Debug Power Control Register
[DBGVCR32_EL2](#): Debug Vector Catch Register
[DBGWCR<n>_EL1](#): Debug Watchpoint Control Registers
[DBGWVR<n>_EL1](#): Debug Watchpoint Value Registers

[DCZID_EL0](#): Data Cache Zero ID register

[DISR_EL1](#): Deferred Interrupt Status Register

[DIT](#): Data Independent Timing

[DLR_EL0](#): Debug Link Register

[DSPSR_EL0](#): Debug Saved Program Status Register

[ELR_EL1](#): Exception Link Register (EL1)

[ELR_EL2](#): Exception Link Register (EL2)

[ELR_EL3](#): Exception Link Register (EL3)

[ERRIDR_EL1](#): Error Record ID Register

[ERRSELR_EL1](#): Error Record Select Register

[ERXADDR_EL1](#): Selected Error Record Address Register

[ERXCTLR_EL1](#): Selected Error Record Control Register

[ERXFR_EL1](#): Selected Error Record Feature Register

[ERXMISC0_EL1](#): Selected Error Record Miscellaneous Register 0

[ERXMISC1_EL1](#): Selected Error Record Miscellaneous Register 1

[ERXMISC2_EL1](#): Selected Error Record Miscellaneous Register 2

[ERXMISC3_EL1](#): Selected Error Record Miscellaneous Register 3

[ERXPFGCDN_EL1](#): Selected Pseudo-fault Generation Countdown register

[ERXPFGCTL_EL1](#): Selected Pseudo-fault Generation Control register

[ERXPFGF_EL1](#): Selected Pseudo-fault Generation Feature register

[ERXSTATUS_EL1](#): Selected Error Record Primary Status Register

[ESR_EL1](#): Exception Syndrome Register (EL1)

[ESR_EL2](#): Exception Syndrome Register (EL2)

[ESR_EL3](#): Exception Syndrome Register (EL3)

[FAR_EL1](#): Fault Address Register (EL1)

[FAR_EL2](#): Fault Address Register (EL2)

[FAR_EL3](#): Fault Address Register (EL3)

[FPCR](#): Floating-point Control Register

[FPEXC32_EL2](#): Floating-Point Exception Control register

[FPSR](#): Floating-point Status Register

[GCR_EL1](#): Tag Control Register.

[GMID_EL1](#): Multiple tag transfer ID register

[GPCCR_EL3](#): Granule Protection Check Control Register (EL3)

[GPTBR_EL3](#): Granule Protection Table Base Register

[HACR_EL2](#): Hypervisor Auxiliary Control Register

[HAFGTR_EL2](#): Hypervisor Activity Monitors Fine-Grained Read Trap Register

[HCRX_EL2](#): Extended Hypervisor Configuration Register
[HCR_EL2](#): Hypervisor Configuration Register
[HDFGRTR_EL2](#): Hypervisor Debug Fine-Grained Read Trap Register
[HDFGWTR_EL2](#): Hypervisor Debug Fine-Grained Write Trap Register
[HFGITR_EL2](#): Hypervisor Fine-Grained Instruction Trap Register
[HFGRTR_EL2](#): Hypervisor Fine-Grained Read Trap Register
[HFGWTR_EL2](#): Hypervisor Fine-Grained Write Trap Register
[HPFAR_EL2](#): Hypervisor IPA Fault Address Register
[HSTR_EL2](#): Hypervisor System Trap Register
[ICC_AP0R<n>_EL1](#): Interrupt Controller Active Priorities Group 0 Registers
[ICC_AP1R<n>_EL1](#): Interrupt Controller Active Priorities Group 1 Registers
[ICC_ASGI1R_EL1](#): Interrupt Controller Alias Software Generated Interrupt Group 1 Register
[ICC_BPR0_EL1](#): Interrupt Controller Binary Point Register 0
[ICC_BPR1_EL1](#): Interrupt Controller Binary Point Register 1
[ICC_CTLR_EL1](#): Interrupt Controller Control Register (EL1)
[ICC_CTLR_EL3](#): Interrupt Controller Control Register (EL3)
[ICC_DIR_EL1](#): Interrupt Controller Deactivate Interrupt Register
[ICC_EOIR0_EL1](#): Interrupt Controller End Of Interrupt Register 0
[ICC_EOIR1_EL1](#): Interrupt Controller End Of Interrupt Register 1
[ICC_HPPIR0_EL1](#): Interrupt Controller Highest Priority Pending Interrupt Register 0
[ICC_HPPIR1_EL1](#): Interrupt Controller Highest Priority Pending Interrupt Register 1
[ICC_IAR0_EL1](#): Interrupt Controller Interrupt Acknowledge Register 0
[ICC_IAR1_EL1](#): Interrupt Controller Interrupt Acknowledge Register 1
[ICC_IGRPEN0_EL1](#): Interrupt Controller Interrupt Group 0 Enable register
[ICC_IGRPEN1_EL1](#): Interrupt Controller Interrupt Group 1 Enable register
[ICC_IGRPEN1_EL3](#): Interrupt Controller Interrupt Group 1 Enable register (EL3)
[ICC_NMIAR1_EL1](#): Interrupt Controller Non-maskable Interrupt Acknowledge Register 1
[ICC_PMR_EL1](#): Interrupt Controller Interrupt Priority Mask Register
[ICC_RPR_EL1](#): Interrupt Controller Running Priority Register
[ICC_SGI0R_EL1](#): Interrupt Controller Software Generated Interrupt Group 0 Register
[ICC_SGI1R_EL1](#): Interrupt Controller Software Generated Interrupt Group 1 Register
[ICC_SRE_EL1](#): Interrupt Controller System Register Enable register (EL1)
[ICC_SRE_EL2](#): Interrupt Controller System Register Enable register (EL2)
[ICC_SRE_EL3](#): Interrupt Controller System Register Enable register (EL3)
[ICH_AP0R<n>_EL2](#): Interrupt Controller Hyp Active Priorities Group 0 Registers
[ICH_AP1R<n>_EL2](#): Interrupt Controller Hyp Active Priorities Group 1 Registers

[ICH_EISR_EL2](#): Interrupt Controller End of Interrupt Status Register

[ICH_ELRSR_EL2](#): Interrupt Controller Empty List Register Status Register

[ICH_HCR_EL2](#): Interrupt Controller Hyp Control Register

[ICH_LR<n>_EL2](#): Interrupt Controller List Registers

[ICH_MISR_EL2](#): Interrupt Controller Maintenance Interrupt State Register

[ICH_VMCR_EL2](#): Interrupt Controller Virtual Machine Control Register

[ICH_VTR_EL2](#): Interrupt Controller VGIC Type Register

[ICV_AP0R<n>_EL1](#): Interrupt Controller Virtual Active Priorities Group 0 Registers

[ICV_AP1R<n>_EL1](#): Interrupt Controller Virtual Active Priorities Group 1 Registers

[ICV_BPR0_EL1](#): Interrupt Controller Virtual Binary Point Register 0

[ICV_BPR1_EL1](#): Interrupt Controller Virtual Binary Point Register 1

[ICV_CTLR_EL1](#): Interrupt Controller Virtual Control Register

[ICV_DIR_EL1](#): Interrupt Controller Deactivate Virtual Interrupt Register

[ICV_EOIR0_EL1](#): Interrupt Controller Virtual End Of Interrupt Register 0

[ICV_EOIR1_EL1](#): Interrupt Controller Virtual End Of Interrupt Register 1

[ICV_HPPIR0_EL1](#): Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0

[ICV_HPPIR1_EL1](#): Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1

[ICV_IAR0_EL1](#): Interrupt Controller Virtual Interrupt Acknowledge Register 0

[ICV_IAR1_EL1](#): Interrupt Controller Virtual Interrupt Acknowledge Register 1

[ICV_IGRPEN0_EL1](#): Interrupt Controller Virtual Interrupt Group 0 Enable register

[ICV_IGRPEN1_EL1](#): Interrupt Controller Virtual Interrupt Group 1 Enable register

[ICV_NMIAR1_EL1](#): Interrupt Controller Virtual Non-maskable Interrupt Acknowledge Register 1

[ICV_PMR_EL1](#): Interrupt Controller Virtual Interrupt Priority Mask Register

[ICV_RPR_EL1](#): Interrupt Controller Virtual Running Priority Register

[ID_AA64AFR0_EL1](#): AArch64 Auxiliary Feature Register 0

[ID_AA64AFR1_EL1](#): AArch64 Auxiliary Feature Register 1

[ID_AA64DFR0_EL1](#): AArch64 Debug Feature Register 0

[ID_AA64DFR1_EL1](#): AArch64 Debug Feature Register 1

[ID_AA64ISAR0_EL1](#): AArch64 Instruction Set Attribute Register 0

[ID_AA64ISAR1_EL1](#): AArch64 Instruction Set Attribute Register 1

[ID_AA64ISAR2_EL1](#): AArch64 Instruction Set Attribute Register 2

[ID_AA64MMFR0_EL1](#): AArch64 Memory Model Feature Register 0

[ID_AA64MMFR1_EL1](#): AArch64 Memory Model Feature Register 1

[ID_AA64MMFR2_EL1](#): AArch64 Memory Model Feature Register 2

[ID_AA64PFR0_EL1](#): AArch64 Processor Feature Register 0

[ID_AA64PFR1_EL1](#): AArch64 Processor Feature Register 1

[ID_AA64SMFR0_EL1](#): SME Feature ID register 0

[ID_AA64ZFR0_EL1](#): SVE Feature ID register 0

[ID_AFR0_EL1](#): AArch32 Auxiliary Feature Register 0

[ID_DFR0_EL1](#): AArch32 Debug Feature Register 0

[ID_DFR1_EL1](#): Debug Feature Register 1

[ID_ISAR0_EL1](#): AArch32 Instruction Set Attribute Register 0

[ID_ISAR1_EL1](#): AArch32 Instruction Set Attribute Register 1

[ID_ISAR2_EL1](#): AArch32 Instruction Set Attribute Register 2

[ID_ISAR3_EL1](#): AArch32 Instruction Set Attribute Register 3

[ID_ISAR4_EL1](#): AArch32 Instruction Set Attribute Register 4

[ID_ISAR5_EL1](#): AArch32 Instruction Set Attribute Register 5

[ID_ISAR6_EL1](#): AArch32 Instruction Set Attribute Register 6

[ID_MMFR0_EL1](#): AArch32 Memory Model Feature Register 0

[ID_MMFR1_EL1](#): AArch32 Memory Model Feature Register 1

[ID_MMFR2_EL1](#): AArch32 Memory Model Feature Register 2

[ID_MMFR3_EL1](#): AArch32 Memory Model Feature Register 3

[ID_MMFR4_EL1](#): AArch32 Memory Model Feature Register 4

[ID_MMFR5_EL1](#): AArch32 Memory Model Feature Register 5

[ID_PFR0_EL1](#): AArch32 Processor Feature Register 0

[ID_PFR1_EL1](#): AArch32 Processor Feature Register 1

[ID_PFR2_EL1](#): AArch32 Processor Feature Register 2

[IFSR32_EL2](#): Instruction Fault Status Register (EL2)

[ISR_EL1](#): Interrupt Status Register

[LORC_EL1](#): LORegion Control (EL1)

[LOREA_EL1](#): LORegion End Address (EL1)

[LORID_EL1](#): LORegionID (EL1)

[LORN_EL1](#): LORegion Number (EL1)

[LORSA_EL1](#): LORegion Start Address (EL1)

[MAIR_EL1](#): Memory Attribute Indirection Register (EL1)

[MAIR_EL2](#): Memory Attribute Indirection Register (EL2)

[MAIR_EL3](#): Memory Attribute Indirection Register (EL3)

[MDCCINT_EL1](#): Monitor DCC Interrupt Enable Register

[MDCCSR_EL0](#): Monitor DCC Status Register

[MDCR_EL2](#): Monitor Debug Configuration Register (EL2)

[MDCR_EL3](#): Monitor Debug Configuration Register (EL3)

[MDRAR_EL1](#): Monitor Debug ROM Address Register

[MDSCR_EL1](#): Monitor Debug System Control Register

[MFAR_EL3](#): PA Fault Address Register

[MIDR_EL1](#): Main ID Register

[MPAM0_EL1](#): MPAM0 Register (EL1)

[MPAM1_EL1](#): MPAM1 Register (EL1)

[MPAM2_EL2](#): MPAM2 Register (EL2)

[MPAM3_EL3](#): MPAM3 Register (EL3)

[MPAMHCR_EL2](#): MPAM Hypervisor Control Register (EL2)

[MPAMIDR_EL1](#): MPAM ID Register (EL1)

[MPAMSM_EL1](#): MPAM Streaming Mode Register

[MPAMVPM0_EL2](#): MPAM Virtual PARTID Mapping Register 0

[MPAMVPM1_EL2](#): MPAM Virtual PARTID Mapping Register 1

[MPAMVPM2_EL2](#): MPAM Virtual PARTID Mapping Register 2

[MPAMVPM3_EL2](#): MPAM Virtual PARTID Mapping Register 3

[MPAMVPM4_EL2](#): MPAM Virtual PARTID Mapping Register 4

[MPAMVPM5_EL2](#): MPAM Virtual PARTID Mapping Register 5

[MPAMVPM6_EL2](#): MPAM Virtual PARTID Mapping Register 6

[MPAMVPM7_EL2](#): MPAM Virtual PARTID Mapping Register 7

[MPAMVPMV_EL2](#): MPAM Virtual Partition Mapping Valid Register

[MPIDR_EL1](#): Multiprocessor Affinity Register

[MVFR0_EL1](#): AArch32 Media and VFP Feature Register 0

[MVFR1_EL1](#): AArch32 Media and VFP Feature Register 1

[MVFR2_EL1](#): AArch32 Media and VFP Feature Register 2

[NZCV](#): Condition Flags

[OSDLR_EL1](#): OS Double Lock Register

[OSDTRRX_EL1](#): OS Lock Data Transfer Register, Receive

[OSDTRTX_EL1](#): OS Lock Data Transfer Register, Transmit

[OSECCR_EL1](#): OS Lock Exception Catch Control Register

[OSLAR_EL1](#): OS Lock Access Register

[OSLSR_EL1](#): OS Lock Status Register

[PAN](#): Privileged Access Never

[PAR_EL1](#): Physical Address Register

[PMBIDR_EL1](#): Profiling Buffer ID Register

[PMBLIMITR_EL1](#): Profiling Buffer Limit Address Register

[PMBPTR_EL1](#): Profiling Buffer Write Pointer Register

[PMBSR_EL1](#): Profiling Buffer Status/syndrome Register

[PMCCFILTR_EL0](#): Performance Monitors Cycle Count Filter Register

[PMCCNTR_EL0](#): Performance Monitors Cycle Count Register

[PMCEID0_EL0](#): Performance Monitors Common Event Identification register 0

[PMCEID1_EL0](#): Performance Monitors Common Event Identification register 1

[PMCNTENCLR_EL0](#): Performance Monitors Count Enable Clear register

[PMCNTENSET_EL0](#): Performance Monitors Count Enable Set register

[PMCR_EL0](#): Performance Monitors Control Register

[PMEVCNTR<n>_EL0](#): Performance Monitors Event Count Registers

[PMEVTYPER<n>_EL0](#): Performance Monitors Event Type Registers

[PMINTENCLR_EL1](#): Performance Monitors Interrupt Enable Clear register

[PMINTENSET_EL1](#): Performance Monitors Interrupt Enable Set register

[PMMIR_EL1](#): Performance Monitors Machine Identification Register

[PMOVSLR_EL0](#): Performance Monitors Overflow Flag Status Clear Register

[PMOVSET_EL0](#): Performance Monitors Overflow Flag Status Set register

[PMSCR_EL1](#): Statistical Profiling Control Register (EL1)

[PMSCR_EL2](#): Statistical Profiling Control Register (EL2)

[PMSELR_EL0](#): Performance Monitors Event Counter Selection Register

[PMSEVER_EL1](#): Sampling Event Filter Register

[PMSFCR_EL1](#): Sampling Filter Control Register

[PMSICR_EL1](#): Sampling Interval Counter Register

[PMSIDR_EL1](#): Sampling Profiling ID Register

[PMSIRR_EL1](#): Sampling Interval Reload Register

[PMSLATFR_EL1](#): Sampling Latency Filter Register

[PMSNEVER_EL1](#): Sampling Inverted Event Filter Register

[PMSWINC_EL0](#): Performance Monitors Software Increment register

[PMUSERENR_EL0](#): Performance Monitors User Enable Register

[PMXEVCNTR_EL0](#): Performance Monitors Selected Event Count Register

[PMXEVTYPER_EL0](#): Performance Monitors Selected Event Type Register

[REVIDR_EL1](#): Revision ID Register

[RGSR_EL1](#): Random Allocation Tag Seed Register.

[RMR_EL1](#): Reset Management Register (EL1)

[RMR_EL2](#): Reset Management Register (EL2)

[RMR_EL3](#): Reset Management Register (EL3)

[RNDR](#): Random Number

[RNDRRS](#): Reseeded Random Number

[RVBAR_EL1](#): Reset Vector Base Address Register (if EL2 and EL3 not implemented)

[RVBAR_EL2](#): Reset Vector Base Address Register (if EL3 not implemented)

[RVBAR_EL3](#): Reset Vector Base Address Register (if EL3 implemented)

[S3_<op1>_<Cn>_<Cm>_<op2>](#): IMPLEMENTATION DEFINED registers

[SCR_EL3](#): Secure Configuration Register

[SCTLR_EL1](#): System Control Register (EL1)

[SCTLR_EL2](#): System Control Register (EL2)

[SCTLR_EL3](#): System Control Register (EL3)

[SCXTNUM_EL0](#): EL0 Read/Write Software Context Number

[SCXTNUM_EL1](#): EL1 Read/Write Software Context Number

[SCXTNUM_EL2](#): EL2 Read/Write Software Context Number

[SCXTNUM_EL3](#): EL3 Read/Write Software Context Number

[SDER32_EL2](#): AArch32 Secure Debug Enable Register

[SDER32_EL3](#): AArch32 Secure Debug Enable Register

[SMCR_EL1](#): SME Control Register (EL1)

[SMCR_EL2](#): SME Control Register (EL2)

[SMCR_EL3](#): SME Control Register (EL3)

[SMIDR_EL1](#): Streaming Mode Identification Register

[SMPRMAP_EL2](#): Streaming Mode Priority Mapping Register

[SMPRI_EL1](#): Streaming Mode Priority Register

[SPSel](#): Stack Pointer Select

[SPSR_abt](#): Saved Program Status Register (Abort mode)

[SPSR_EL1](#): Saved Program Status Register (EL1)

[SPSR_EL2](#): Saved Program Status Register (EL2)

[SPSR_EL3](#): Saved Program Status Register (EL3)

[SPSR_fiq](#): Saved Program Status Register (FIQ mode)

[SPSR_irq](#): Saved Program Status Register (IRQ mode)

[SPSR_und](#): Saved Program Status Register (Undefined mode)

[SP_EL0](#): Stack Pointer (EL0)

[SP_EL1](#): Stack Pointer (EL1)

[SP_EL2](#): Stack Pointer (EL2)

[SP_EL3](#): Stack Pointer (EL3)

[SSBS](#): Speculative Store Bypass Safe

[SVCR](#): Streaming Vector Control Register

[TCO](#): Tag Check Override

[TCR_EL1](#): Translation Control Register (EL1)

[TCR_EL2](#): Translation Control Register (EL2)

[TCR_EL3](#): Translation Control Register (EL3)
[TFSRE0_EL1](#): Tag Fault Status Register (EL0).
[TFSR_EL1](#): Tag Fault Status Register (EL1)
[TFSR_EL2](#): Tag Fault Status Register (EL2)
[TFSR_EL3](#): Tag Fault Status Register (EL3)
[TPIDR2_EL0](#): EL0 Read/Write Software Thread ID Register 2
[TPIDRRO_EL0](#): EL0 Read-Only Software Thread ID Register
[TPIDR_EL0](#): EL0 Read/Write Software Thread ID Register
[TPIDR_EL1](#): EL1 Software Thread ID Register
[TPIDR_EL2](#): EL2 Software Thread ID Register
[TPIDR_EL3](#): EL3 Software Thread ID Register
[TRBBASER_EL1](#): Trace Buffer Base Address Register
[TRBIDR_EL1](#): Trace Buffer ID Register
[TRBLIMITR_EL1](#): Trace Buffer Limit Address Register
[TRBMAR_EL1](#): Trace Buffer Memory Attribute Register
[TRBPTR_EL1](#): Trace Buffer Write Pointer Register
[TRBSR_EL1](#): Trace Buffer Status/syndrome Register
[TRBTRG_EL1](#): Trace Buffer Trigger Counter Register
[TRCACATR<n>](#): Address Comparator Access Type Register <n>
[TRCACVR<n>](#): Address Comparator Value Register <n>
[TRCAUTHSTATUS](#): Authentication Status Register
[TRCAUXCTLR](#): Auxiliary Control Register
[TRCBBCTLR](#): Branch Broadcast Control Register
[TRCCCCTLR](#): Cycle Count Control Register
[TRCCIDCCTLR0](#): Context Identifier Comparator Control Register 0
[TRCCIDCCTLR1](#): Context Identifier Comparator Control Register 1
[TRCCIDCVR<n>](#): Context Identifier Comparator Value Registers <n>
[TRCCLAIMCLR](#): Claim Tag Clear Register
[TRCCLAIMSET](#): Claim Tag Set Register
[TRCCNTCTLR<n>](#): Counter Control Register <n>
[TRCCNTRLDVR<n>](#): Counter Reload Value Register <n>
[TRCCNTVR<n>](#): Counter Value Register <n>
[TRCCONFIGR](#): Trace Configuration Register
[TRCDEVARCH](#): Device Architecture Register
[TRCDEVID](#): Device Configuration Register
[TRCEVENTCTL0R](#): Event Control 0 Register

[TRCEVENTCTL1R](#): Event Control 1 Register

[TRCEXTINSELR<n>](#): External Input Select Register <n>

[TRCIDR0](#): ID Register 0

[TRCIDR1](#): ID Register 1

[TRCIDR10](#): ID Register 10

[TRCIDR11](#): ID Register 11

[TRCIDR12](#): ID Register 12

[TRCIDR13](#): ID Register 13

[TRCIDR2](#): ID Register 2

[TRCIDR3](#): ID Register 3

[TRCIDR4](#): ID Register 4

[TRCIDR5](#): ID Register 5

[TRCIDR6](#): ID Register 6

[TRCIDR7](#): ID Register 7

[TRCIDR8](#): ID Register 8

[TRCIDR9](#): ID Register 9

[TRCIMSPEC0](#): IMP DEF Register 0

[TRCIMSPEC<n>](#): IMP DEF Register <n>

[TRCOSLSR](#): Trace OS Lock Status Register

[TRCPRGCTLR](#): Programming Control Register

[TRCQCTLR](#): Q Element Control Register

[TRCRSCTLR<n>](#): Resource Selection Control Register <n>

[TRCRSR](#): Resources Status Register

[TRCSEQEVR<n>](#): Sequencer State Transition Control Register <n>

[TRCSEQRSTEV](#): Sequencer Reset Control Register

[TRCSEQSTR](#): Sequencer State Register

[TRCSSCCR<n>](#): Single-shot Comparator Control Register <n>

[TRCSSCSR<n>](#): Single-shot Comparator Control Status Register <n>

[TRCSSPCICR<n>](#): Single-shot Processing Element Comparator Input Control Register <n>

[TRCSTALLCTLR](#): Stall Control Register

[TRCSTATR](#): Trace Status Register

[TRCSYNCP](#): Synchronization Period Register

[TRCTRACEIDR](#): Trace ID Register

[TRCTSCTLR](#): Timestamp Control Register

[TRCVICTLR](#): ViewInst Main Control Register

[TRCVIIECTLR](#): ViewInst Include/Exclude Control Register

[TRCVIPCSSCTLR](#): ViewInst Start/Stop PE Comparator Control Register

[TRCVISSCTLR](#): ViewInst Start/Stop Control Register

[TRCVMIDCCTLR0](#): Virtual Context Identifier Comparator Control Register 0

[TRCVMIDCCTLR1](#): Virtual Context Identifier Comparator Control Register 1

[TRCVMIDCVR<n>](#): Virtual Context Identifier Comparator Value Register <n>

[TRFCR_EL1](#): Trace Filter Control Register (EL1)

[TRFCR_EL2](#): Trace Filter Control Register (EL2)

[TTBR0_EL1](#): Translation Table Base Register 0 (EL1)

[TTBR0_EL2](#): Translation Table Base Register 0 (EL2)

[TTBR0_EL3](#): Translation Table Base Register 0 (EL3)

[TTBR1_EL1](#): Translation Table Base Register 1 (EL1)

[TTBR1_EL2](#): Translation Table Base Register 1 (EL2)

[UAO](#): User Access Override

[VBAR_EL1](#): Vector Base Address Register (EL1)

[VBAR_EL2](#): Vector Base Address Register (EL2)

[VBAR_EL3](#): Vector Base Address Register (EL3)

[VDISR_EL2](#): Virtual Deferred Interrupt Status Register

[VMPIDR_EL2](#): Virtualization Multiprocessor ID Register

[VNCR_EL2](#): Virtual Nested Control Register

[VPIDR_EL2](#): Virtualization Processor ID Register

[VSESR_EL2](#): Virtual SError Exception Syndrome Register

[VSTCR_EL2](#): Virtualization Secure Translation Control Register

[VSTTBR_EL2](#): Virtualization Secure Translation Table Base Register

[VTCR_EL2](#): Virtualization Translation Control Register

[VTTBR_EL2](#): Virtualization Translation Table Base Register

[ZCR_EL1](#): SVE Control Register (EL1)

[ZCR_EL2](#): SVE Control Register (EL2)

[ZCR_EL3](#): SVE Control Register (EL3)

30/09/2021 14:57

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AArch64 System Instructions

[AT S12E0R](#): Address Translate Stages 1 and 2 EL0 Read
[AT S12E0W](#): Address Translate Stages 1 and 2 EL0 Write
[AT S12E1R](#): Address Translate Stages 1 and 2 EL1 Read
[AT S12E1W](#): Address Translate Stages 1 and 2 EL1 Write
[AT S1E0R](#): Address Translate Stage 1 EL0 Read
[AT S1E0W](#): Address Translate Stage 1 EL0 Write
[AT S1E1R](#): Address Translate Stage 1 EL1 Read
[AT S1E1RP](#): Address Translate Stage 1 EL1 Read PAN
[AT S1E1W](#): Address Translate Stage 1 EL1 Write
[AT S1E1WP](#): Address Translate Stage 1 EL1 Write PAN
[AT S1E2R](#): Address Translate Stage 1 EL2 Read
[AT S1E2W](#): Address Translate Stage 1 EL2 Write
[AT S1E3R](#): Address Translate Stage 1 EL3 Read
[AT S1E3W](#): Address Translate Stage 1 EL3 Write
[BRB IALL](#): Invalidate the Branch Record Buffer
[BRB INJ](#): Branch Record Injection into the Branch Record Buffer
[CFP RCTX](#): Control Flow Prediction Restriction by Context
[CPP RCTX](#): Cache Prefetch Prediction Restriction by Context
[DC CGDSW](#): Clean of Data and Allocation Tags by Set/Way
[DC CGDVAC](#): Clean of Data and Allocation Tags by VA to PoC
[DC CGDVADP](#): Clean of Data and Allocation Tags by VA to PoDP
[DC CGDVAP](#): Clean of Data and Allocation Tags by VA to PoP
[DC CGSW](#): Clean of Allocation Tags by Set/Way
[DC CGVAC](#): Clean of Allocation Tags by VA to PoC
[DC CGVADP](#): Clean of Allocation Tags by VA to PoDP
[DC CGVAP](#): Clean of Allocation Tags by VA to PoP
[DC CIGDPAPA](#): Clean and Invalidate of Data and Allocation Tags by PA to PoPA
[DC CIGDSW](#): Clean and Invalidate of Data and Allocation Tags by Set/Way
[DC CIGDVAC](#): Clean and Invalidate of Data and Allocation Tags by VA to PoC
[DC CIGSW](#): Clean and Invalidate of Allocation Tags by Set/Way
[DC CIGVAC](#): Clean and Invalidate of Allocation Tags by VA to PoC
[DC CIPAPA](#): Data or unified Cache line Clean and Invalidate by PA to PoPA
[DC CISW](#): Data or unified Cache line Clean and Invalidate by Set/Way
[DC CIVAC](#): Data or unified Cache line Clean and Invalidate by VA to PoC

[DC CSW](#): Data or unified Cache line Clean by Set/Way

[DC CVAC](#): Data or unified Cache line Clean by VA to PoC

[DC CVADP](#): Data or unified Cache line Clean by VA to PoDP

[DC CVAP](#): Data or unified Cache line Clean by VA to PoP

[DC CVAU](#): Data or unified Cache line Clean by VA to PoU

[DC GVA](#): Data Cache set Allocation Tag by VA

[DC GZVA](#): Data Cache set Allocation Tags and Zero by VA

[DC IGDSW](#): Invalidate of Data and Allocation Tags by Set/Way

[DC IGDVAC](#): Invalidate of Data and Allocation Tags by VA to PoC

[DC IGSW](#): Invalidate of Allocation Tags by Set/Way

[DC IGVAC](#): Invalidate of Allocation Tags by VA to PoC

[DC ISW](#): Data or unified Cache line Invalidate by Set/Way

[DC IVAC](#): Data or unified Cache line Invalidate by VA to PoC

[DC ZVA](#): Data Cache Zero by VA

[DVP RCTX](#): Data Value Prediction Restriction by Context

[IC IALLU](#): Instruction Cache Invalidate All to PoU

[IC IALLUIS](#): Instruction Cache Invalidate All to PoU, Inner Shareable

[IC IVAU](#): Instruction Cache line Invalidate by VA to PoU

[SYS S1_<op1>_<Cn>_<Cm>_<op2>, SYSL S1_<op1>_<Cn>_<Cm>_<op2>](#): IMPLEMENTATION DEFINED maintenance instructions

[TLBI ALLE1, TLBI ALLE1NXS](#): TLB Invalidate All, EL1

[TLBI ALLE1IS, TLBI ALLE1ISNXS](#): TLB Invalidate All, EL1, Inner Shareable

[TLBI ALLE1OS, TLBI ALLE1OSNXS](#): TLB Invalidate All, EL1, Outer Shareable

[TLBI ALLE2, TLBI ALLE2NXS](#): TLB Invalidate All, EL2

[TLBI ALLE2IS, TLBI ALLE2ISNXS](#): TLB Invalidate All, EL2, Inner Shareable

[TLBI ALLE2OS, TLBI ALLE2OSNXS](#): TLB Invalidate All, EL2, Outer Shareable

[TLBI ALLE3, TLBI ALLE3NXS](#): TLB Invalidate All, EL3

[TLBI ALLE3IS, TLBI ALLE3ISNXS](#): TLB Invalidate All, EL3, Inner Shareable

[TLBI ALLE3OS, TLBI ALLE3OSNXS](#): TLB Invalidate All, EL3, Outer Shareable

[TLBI ASIDE1, TLBI ASIDE1NXS](#): TLB Invalidate by ASID, EL1

[TLBI ASIDE1IS, TLBI ASIDE1ISNXS](#): TLB Invalidate by ASID, EL1, Inner Shareable

[TLBI ASIDE1OS, TLBI ASIDE1OSNXS](#): TLB Invalidate by ASID, EL1, Outer Shareable

[TLBI IPAS2E1, TLBI IPAS2E1NXS](#): TLB Invalidate by Intermediate Physical Address, Stage 2, EL1

[TLBI IPAS2E1IS, TLBI IPAS2E1ISNXS](#): TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

[TLBI IPAS2E1OS, TLBI IPAS2E1OSNXS](#): TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

[TLBI IPAS2LE1, TLBI IPAS2LE1NXS](#): TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

[TLBI IPAS2LE1IS, TLBI IPAS2LE1ISNXS](#): TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

[TLBI IPAS2LE1OS, TLBI IPAS2LE1OSNXS](#): TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

[TLBI PAALL](#): TLB Invalidate GPT Information by PA, All Entries, Local

[TLBI PAALLOS](#): TLB Invalidate GPT Information by PA, All Entries, Outer Shareable

[TLBI RIPAS2E1, TLBI RIPAS2E1NXS](#): TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1

[TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS](#): TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

[TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS](#): TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

[TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS](#): TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

[TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS](#): TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

[TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS](#): TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

[TLBI RPALOS](#): TLB Range Invalidate GPT Information by PA, Last level, Outer Shareable

[TLBI RPAOS](#): TLB Range Invalidate GPT Information by PA, Outer Shareable

[TLBI RVAAE1, TLBI RVAAE1NXS](#): TLB Range Invalidate by VA, All ASID, EL1

[TLBI RVAAE1IS, TLBI RVAAE1ISNXS](#): TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable

[TLBI RVAAE1OS, TLBI RVAAE1OSNXS](#): TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable

[TLBI RVAALE1, TLBI RVAALE1NXS](#): TLB Range Invalidate by VA, All ASID, Last level, EL1

[TLBI RVAALE1IS, TLBI RVAALE1ISNXS](#): TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

[TLBI RVAALE1OS, TLBI RVAALE1OSNXS](#): TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

[TLBI RVAE1, TLBI RVAE1NXS](#): TLB Range Invalidate by VA, EL1

[TLBI RVAE1IS, TLBI RVAE1ISNXS](#): TLB Range Invalidate by VA, EL1, Inner Shareable

[TLBI RVAE1OS, TLBI RVAE1OSNXS](#): TLB Range Invalidate by VA, EL1, Outer Shareable

[TLBI RVAE2, TLBI RVAE2NXS](#): TLB Range Invalidate by VA, EL2

[TLBI RVAE2IS, TLBI RVAE2ISNXS](#): TLB Range Invalidate by VA, EL2, Inner Shareable

[TLBI RVAE2OS, TLBI RVAE2OSNXS](#): TLB Range Invalidate by VA, EL2, Outer Shareable

[TLBI RVAE3, TLBI RVAE3NXS](#): TLB Range Invalidate by VA, EL3

[TLBI RVAE3IS, TLBI RVAE3ISNXS](#): TLB Range Invalidate by VA, EL3, Inner Shareable

[TLBI RVAE3OS, TLBI RVAE3OSNXS](#): TLB Range Invalidate by VA, EL3, Outer Shareable

[TLBI RVALE1, TLBI RVALE1NXS](#): TLB Range Invalidate by VA, Last level, EL1

[TLBI RVALE1IS, TLBI RVALE1ISNXS](#): TLB Range Invalidate by VA, Last level, EL1, Inner Shareable

[TLBI RVALE1OS, TLBI RVALE1OSNXS](#): TLB Range Invalidate by VA, Last level, EL1, Outer Shareable

[TLBI RVALE2, TLBI RVALE2NXS](#): TLB Range Invalidate by VA, Last level, EL2

[TLBI RVALE2IS, TLBI RVALE2ISNXS](#): TLB Range Invalidate by VA, Last level, EL2, Inner Shareable

[TLBI RVALE2OS, TLBI RVALE2OSNXS](#): TLB Range Invalidate by VA, Last level, EL2, Outer Shareable

[TLBI RVALE3, TLBI RVALE3NXS](#): TLB Range Invalidate by VA, Last level, EL3

[TLBI RVALE3IS, TLBI RVALE3ISNXS](#): TLB Range Invalidate by VA, Last level, EL3, Inner Shareable

[TLBI RVALE3OS, TLBI RVALE3OSNXS](#): TLB Range Invalidate by VA, Last level, EL3, Outer Shareable

[TLBI VAAE1, TLBI VAAE1NXS](#): TLB Invalidate by VA, All ASID, EL1

[TLBI VAAE1IS, TLBI VAAE1ISNXS](#): TLB Invalidate by VA, All ASID, EL1, Inner Shareable

[TLBI VAAE1OS, TLBI VAAE1OSNXS](#): TLB Invalidate by VA, All ASID, EL1, Outer Shareable

[TLBI VAALE1, TLBI VAALE1NXS](#): TLB Invalidate by VA, All ASID, Last level, EL1

[TLBI VAALE1IS, TLBI VAALE1ISNXS](#): TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

[TLBI VAALE1OS, TLBI VAALE1OSNXS](#): TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

[TLBI VAE1, TLBI VAE1NXS](#): TLB Invalidate by VA, EL1

[TLBI VAE1IS, TLBI VAE1ISNXS](#): TLB Invalidate by VA, EL1, Inner Shareable

[TLBI VAE1OS, TLBI VAE1OSNXS](#): TLB Invalidate by VA, EL1, Outer Shareable

[TLBI VAE2, TLBI VAE2NXS](#): TLB Invalidate by VA, EL2

[TLBI VAE2IS, TLBI VAE2ISNXS](#): TLB Invalidate by VA, EL2, Inner Shareable

[TLBI VAE2OS, TLBI VAE2OSNXS](#): TLB Invalidate by VA, EL2, Outer Shareable

[TLBI VAE3, TLBI VAE3NXS](#): TLB Invalidate by VA, EL3

[TLBI VAE3IS, TLBI VAE3ISNXS](#): TLB Invalidate by VA, EL3, Inner Shareable

[TLBI VAE3OS, TLBI VAE3OSNXS](#): TLB Invalidate by VA, EL3, Outer Shareable

[TLBI VALE1, TLBI VALE1NXS](#): TLB Invalidate by VA, Last level, EL1

[TLBI VALE1IS, TLBI VALE1ISNXS](#): TLB Invalidate by VA, Last level, EL1, Inner Shareable

[TLBI VALE1OS, TLBI VALE1OSNXS](#): TLB Invalidate by VA, Last level, EL1, Outer Shareable

[TLBI VALE2, TLBI VALE2NXS](#): TLB Invalidate by VA, Last level, EL2

[TLBI VALE2IS, TLBI VALE2ISNXS](#): TLB Invalidate by VA, Last level, EL2, Inner Shareable

[TLBI VALE2OS, TLBI VALE2OSNXS](#): TLB Invalidate by VA, Last level, EL2, Outer Shareable

[TLBI VALE3, TLBI VALE3NXS](#): TLB Invalidate by VA, Last level, EL3

[TLBI VALE3IS, TLBI VALE3ISNXS](#): TLB Invalidate by VA, Last level, EL3, Inner Shareable

[TLBI VALE3OS, TLBI VALE3OSNXS](#): TLB Invalidate by VA, Last level, EL3, Outer Shareable

[TLBI VMALLE1, TLBI VMALLE1NXS](#): TLB Invalidate by VMID, All at stage 1, EL1

[TLBI VMALLE1IS, TLBI VMALLE1ISNXS](#): TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable

[TLBI VMALLE1OS, TLBI VMALLE1OSNXS](#): TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable

[TLBI VMALLS12E1, TLBI VMALLS12E1NXS](#): TLB Invalidate by VMID, All at Stage 1 and 2, EL1

[TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS](#): TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable

[TLBI VMALLS12E1OS, TLBI VMALLS12E1OSNXS](#): TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable

ACCDATA_EL1, Accelerator Data

The ACCDATA_EL1 characteristics are:

Purpose

Holds the lower 32 bits of the data that is stored by an ST64BV0, Single-copy atomic 64-byte EL0 store instruction.

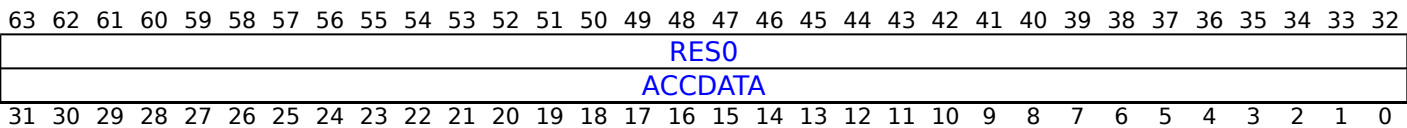
Configuration

This register is present only when FEAT_LS64_ACCDATA is implemented. Otherwise, direct accesses to ACCDATA_EL1 are UNDEFINED.

Attributes

ACCDATA_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

ACCDATA, bits [31:0]

Accelerator Data field. Holds bits[31:0] of the data that is stored by an ST64BV0 instruction.

Accessing ACCDATA_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ACCDATA_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ADEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.nACCDATA_EL1 == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ADEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ACCDATA_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ADEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ADEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ACCDATA_EL1;
elsif PSTATE.EL == EL3 then
    return ACCDATA_EL1;

```

MSR ACCDATA_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ADEn == '0' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.nACCDATA_EL1 == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ADEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ACCDATA_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ADEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ADEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ACCDATA_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ACCDATA_EL1 = X[t];

```


ACTLR_EL1, Auxiliary Control Register (EL1)

The ACTLR_EL1 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED configuration and control options for execution at EL1 and EL0.

Note

Arm recommends the contents of this register have no effect on the PE when [HCR_EL2](#).{E2H, TGE} is {1, 1}, and instead the configuration and control fields are provided by the [ACTLR_EL2](#) register. This avoids the need for software to manage the contents of these register when switching between a Guest OS and a Host OS.

Configuration

AArch64 System register ACTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ACTLR\[31:0\]](#).

AArch64 System register ACTLR_EL1 bits [63:32] are architecturally mapped to AArch32 System register [ACTLR2\[31:0\]](#).

Attributes

ACTLR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ACTLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ACTLR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TACR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x118];
    else
        return ACTLR_EL1;
elsif PSTATE.EL == EL2 then
    return ACTLR_EL1;
elsif PSTATE.EL == EL3 then
    return ACTLR_EL1;

```

MSR ACTLR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TACR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        NVMem[0x118] = X[t];
    else
        ACTLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    ACTLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ACTLR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ACTLR_EL2, Auxiliary Control Register (EL2)

The ACTLR_EL2 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED configuration and control options for EL2.

Note

Arm recommends the contents of this register are updated to apply to EL0 when [HCR_EL2](#).{E2H, TGE} is {1, 1}, gaining configuration and control fields from the [ACTLR_EL1](#). This avoids the need for software to manage the contents of these register when switching between a Guest OS and a Host OS.

Configuration

AArch64 System register ACTLR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HACTLR\[31:0\]](#).

AArch64 System register ACTLR_EL2 bits [63:32] are architecturally mapped to AArch32 System register [HACTLR2\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ACTLR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ACTLR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ACTLR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return ACTLR_EL2;
elsif PSTATE.EL == EL3 then
    return ACTLR_EL2;

```

MSR ACTLR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ACTLR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    ACTLR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ACTLR_EL3, Auxiliary Control Register (EL3)

The ACTLR_EL3 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED configuration and control options for EL3.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to ACTLR_EL3 are UNDEFINED.

Attributes

ACTLR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ACTLR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ACTLR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0000 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return ACTLR_EL3;
```

MSR ACTLR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0000 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    ACTLR_EL3 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AFSR0_EL1, Auxiliary Fault Status Register 0 (EL1)

The AFSR0_EL1 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL1.

Configuration

AArch64 System register AFSR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ADFSR\[31:0\]](#).

Attributes

AFSR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AFSR0_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic AFSR0_EL1 or AFSR0_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AFSR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.AFSR0_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x128];
    else
        return AFSR0_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AFSR0_EL2;
    else
        return AFSR0_EL1;
elsif PSTATE.EL == EL3 then
    return AFSR0_EL1;

```

MSR AFSR0_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.AFSR0_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x128] = X[t];
    else
        AFSR0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AFSR0_EL2 = X[t];
    else
        AFSR0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    AFSR0_EL1 = X[t];

```

MRS <Xt>, AFSR0_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0101 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x128];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AFSR0_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return AFSR0_EL1;
    else
        UNDEFINED;

```

MSR AFSR0_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0101 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x128] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AFSR0_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        AFSR0_EL1 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AFSR0_EL2, Auxiliary Fault Status Register 0 (EL2)

The AFSR0_EL2 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL2.

Configuration

AArch64 System register AFSR0_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HADFSTR\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

AFSR0_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AFSR0_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic AFSR0_EL2 or AFSR0_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AFSR0_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return AFSR0_EL2;
elsif PSTATE.EL == EL3 then
    return AFSR0_EL2;

```

MSR AFSR0_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AFSR0_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    AFSR0_EL2 = X[t];

```

MRS <Xt>, AFSR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.AFSR0_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x128];
    else
        return AFSR0_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AFSR0_EL2;
    else
        return AFSR0_EL1;
elsif PSTATE.EL == EL3 then
    return AFSR0_EL1;

```

MSR AFSR0_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.AFSR0_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x128] = X[t];
    else
        AFSR0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AFSR0_EL2 = X[t];
    else
        AFSR0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    AFSR0_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AFSR0_EL3, Auxiliary Fault Status Register 0 (EL3)

The AFSR0_EL3 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL3.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to AFSR0_EL3 are UNDEFINED.

Attributes

AFSR0_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AFSR0_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AFSR0_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0101 | 0b0001 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return AFSR0_EL3;
```

MSR AFSR0_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0101 | 0b0001 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AFSR0_EL3 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AFSR1_EL1, Auxiliary Fault Status Register 1 (EL1)

The AFSR1_EL1 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL1.

Configuration

AArch64 System register AFSR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register [AIFSR\[31:0\]](#).

Attributes

AFSR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AFSR1_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic AFSR1_EL1 or AFSR1_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AFSR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.AFSR1_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x130];
    else
        return AFSR1_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AFSR1_EL2;
    else
        return AFSR1_EL1;
elsif PSTATE.EL == EL3 then
    return AFSR1_EL1;

```

MSR AFSR1_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.AFSR1_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x130] = X[t];
    else
        AFSR1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AFSR1_EL2 = X[t];
    else
        AFSR1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    AFSR1_EL1 = X[t];

```

MRS <Xt>, AFSR1_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0101 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x130];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AFSR1_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return AFSR1_EL1;
    else
        UNDEFINED;

```

MSR AFSR1_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0101 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x130] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AFSR1_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        AFSR1_EL1 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AFSR1_EL2, Auxiliary Fault Status Register 1 (EL2)

The AFSR1_EL2 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL2.

Configuration

AArch64 System register AFSR1_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HIAFSR\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

AFSR1_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AFSR1_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic AFSR1_EL2 or AFSR1_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AFSR1_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return AFSR1_EL2;
elsif PSTATE.EL == EL3 then
    return AFSR1_EL2;

```

MSR AFSR1_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AFSR1_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    AFSR1_EL2 = X[t];

```

MRS <Xt>, AFSR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.AFSR1_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x130];
    else
        return AFSR1_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AFSR1_EL2;
    else
        return AFSR1_EL1;
elsif PSTATE.EL == EL3 then
    return AFSR1_EL1;

```

MSR AFSR1_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.AFSR1_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x130] = X[t];
    else
        AFSR1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AFSR1_EL2 = X[t];
    else
        AFSR1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    AFSR1_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AFSR1_EL3, Auxiliary Fault Status Register 1 (EL3)

The AFSR1_EL3 characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for exceptions taken to EL3.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to AFSR1_EL3 are UNDEFINED.

Attributes

AFSR1_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AFSR1_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AFSR1_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0101 | 0b0001 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return AFSR1_EL3;
```

MSR AFSR1_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0101 | 0b0001 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AFSR1_EL3 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AIDR_EL1, Auxiliary ID Register

The AIDR_EL1 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED identification information.

The value of this register must be interpreted in conjunction with the value of [MIDR_EL1](#).

Configuration

AArch64 System register AIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [AIDR\[31:0\]](#).

Attributes

AIDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing AIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b001 | 0b0000 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.AIDR_EL1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return AIDR_EL1;
    elseif PSTATE.EL == EL2 then
        return AIDR_EL1;
    elseif PSTATE.EL == EL3 then
        return AIDR_EL1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ALLINT, All Interrupt Mask Bit

The ALLINT characteristics are:

Purpose

Allows access to the all interrupt mask bit.

Configuration

This register is present only when FEAT_NMI is implemented. Otherwise, direct accesses to ALLINT are UNDEFINED.

Attributes

ALLINT is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | ALLINT | | RES0 | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | |

Bits [63:14]

Reserved, RES0.

ALLINT, bit [13]

All interrupt mask.

| ALLINT | Meaning |
|--------|--|
| 0b0 | When SCTL_ELx.NMI is 1 and execution is at ELx, an IRQ or FIQ interrupt with Superpriority that is targeted to ELx is not masked by PSTATE.I or PSTATE.F, respectively, unless both PSTATE.SP and SCTL_ELx.SPINTMASK are 1. An IRQ or FIQ interrupt without Superpriority that is targeted to ELx is masked. |
| 0b1 | When SCTL_ELx.NMI is 1 and execution is at ELx, an IRQ or FIQ interrupt that is targeted to ELx is masked by PSTATE.I or PSTATE.F, respectively, regardless of Superpriority. |

When executing at EL0 and SCTL_ELx.NMI is 1, if an IRQ or FIQ interrupt targeted to ELx is masked by PSTATE.I or PSTATE.F, the mask applies only to IRQ or FIQ interrupts without Superpriority. IRQ or FIQ interrupts with Superpriority are not masked.

The value of this bit is set to the inverse value in the SCTL_ELx.SPINTMASK field on taking an exception to ELx.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [12:0]

Reserved, RES0.

Accessing ALLINT

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ALLINT

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0011 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    return Zeros(50):PSTATE.ALLINT:Zeros(13);
elsif PSTATE.EL == EL2 then
    return Zeros(50):PSTATE.ALLINT:Zeros(13);
elsif PSTATE.EL == EL3 then
    return Zeros(50):PSTATE.ALLINT:Zeros(13);
```

MSR ALLINT, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0011 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && IsHCRXEL2Enabled() && HCRX_EL2.TALLINT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        PSTATE.ALLINT = X[t]<13>;
elsif PSTATE.EL == EL2 then
    PSTATE.ALLINT = X[t]<13>;
elsif PSTATE.EL == EL3 then
    PSTATE.ALLINT = X[t]<13>;
```

MSR ALLINT, #<imm>

| op0 | op1 | CRn | op2 |
|------|-------|--------|-------|
| 0b00 | 0b001 | 0b0100 | 0b000 |

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMAIR_EL1, Auxiliary Memory Attribute Indirection Register (EL1)

The AMAIR_EL1 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by [MAIR_EL1](#).

Configuration

AArch64 System register AMAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [AMAIRO\[31:0\]](#).

AArch64 System register AMAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register [AMAIR1\[31:0\]](#).

Attributes

AMAIR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

AMAIR_EL1 is permitted to be cached in a TLB.

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMAIR_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic AMAIR_EL1 or AMAIR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMAIR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.AMAIR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x148];
    else
        return AMAIR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AMAIR_EL2;
    else
        return AMAIR_EL1;
elsif PSTATE.EL == EL3 then
    return AMAIR_EL1;
    
```

MSR AMAIR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.AMAIR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x148] = X[t];
    else
        AMAIR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AMAIR_EL2 = X[t];
    else
        AMAIR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    AMAIR_EL1 = X[t];
    
```

MRS <Xt>, AMAIR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x148];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AMAIR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return AMAIR_EL1;
    else
        UNDEFINED;

```

MSR AMAIR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x148] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AMAIR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        AMAIR_EL1 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMAIR_EL2, Auxiliary Memory Attribute Indirection Register (EL2)

The AMAIR_EL2 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by [MAIR_EL2](#).

Configuration

AArch64 System register AMAIR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HMAIRO\[31:0\]](#).

AArch64 System register AMAIR_EL2 bits [63:32] are architecturally mapped to AArch32 System register [HMAIR1\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

AMAIR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

AMAIR_EL2 is permitted to be cached in a TLB.

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMAIR_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic AMAIR_EL2 or AMAIR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMAIR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return AMAIR_EL2;
elsif PSTATE.EL == EL3 then
    return AMAIR_EL2;

```

MSR AMAIR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AMAIR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    AMAIR_EL2 = X[t];

```

MRS <Xt>, AMAIR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.AMAIR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x148];
    else
        return AMAIR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return AMAIR_EL2;
    else
        return AMAIR_EL1;
elsif PSTATE.EL == EL3 then
    return AMAIR_EL1;

```

MSR AMAIR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.AMAIR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x148] = X[t];
    else
        AMAIR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AMAIR_EL2 = X[t];
    else
        AMAIR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    AMAIR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMAIR_EL3, Auxiliary Memory Attribute Indirection Register (EL3)

The AMAIR_EL3 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by [MAIR_EL3](#).

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to AMAIR_EL3 are UNDEFINED.

Attributes

AMAIR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

AMAIR_EL3 is permitted to be cached in a TLB.

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMAIR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMAIR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return AMAIR_EL3;

```

MSR AMAIR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AMAIR_EL3 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCFGR_EL0, Activity Monitors Configuration Register

The AMCFGR_EL0 characteristics are:

Purpose

Global configuration register for the activity monitors.

Provides information on supported features, the number of counter groups implemented, the total number of activity monitor event counters implemented, and the size of the counters. AMCFGR_EL0 is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch64 System register AMCFGR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [AMCFGR\[31:0\]](#).

AArch64 System register AMCFGR_EL0 bits [31:0] are architecturally mapped to External register [AMCFGR\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCFGR_EL0 are UNDEFINED.

Attributes

AMCFGR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|------|----|----|----|------|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|---|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| NCG | | | | RES0 | | | | HDBG | | | | RAZ | | | | | | | | | | | | SIZE | | | | | | | | N | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

Bits [63:32]

Reserved, RES0.

NCG, bits [31:28]

Defines the number of counter groups.

The number of implemented counter groups is [AMCFGR_EL0.NCG + 1].

If the number of implemented auxiliary activity monitor event counters is zero, this field has a value of 0b0000. Otherwise, this field has a value of 0b0001.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [27:25]

Reserved, RES0.

HDBG, bit [24]

Halt-on-debug supported.

This feature must be supported, and so this bit is 0b1.

| HDBG | Meaning |
|------|---|
| 0b0 | AMCR_EL0 .HDBG is RES0. |
| 0b1 | AMCR_EL0 .HDBG is read/write. |

Access to this field is **RO**.

Bits [23:14]

Reserved, RAZ.

SIZE, bits [13:8]

Defines the size of activity monitor event counters.

The size of the activity monitor event counters implemented by the activity monitors Extension is [AMCFGR_EL0.SIZE + 1].

Note

Software also uses this field to determine the spacing of counters in the memory-map. From Armv8, the counters are at doubleword-aligned addresses.

Reads as 0b111111.

Access to this field is **RO**.

N, bits [7:0]

Defines the number of activity monitor event counters.

The total number of counters implemented in all groups by the Activity Monitors Extension is [AMCFGR_EL0.N + 1].

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMCFGR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMCFGR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCFGR_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCFGR_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCFGR_EL0;
    elsif PSTATE.EL == EL3 then
        return AMCFGR_EL0;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCG1IDR_EL0, Activity Monitors Counter Group 1 Identification Register

The AMCG1IDR_EL0 characteristics are:

Purpose

Defines which auxiliary counters are implemented, and which of them have a corresponding virtual offset register, [AMEVCNTVOFF1<n>_EL2](#) implemented.

Configuration

This register is present only when FEAT_AMUv1p1 is implemented. Otherwise, direct accesses to AMCG1IDR_EL0 are UNDEFINED.

Attributes

AMCG1IDR_EL0 is a 64-bit register.

Field descriptions

| | | | | | |
|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|
| 63 | 62 | 61 | 60 | 59 | |
| AMEVCNTVOFF115_EL2 | AMEVCNTVOFF114_EL2 | AMEVCNTVOFF113_EL2 | AMEVCNTVOFF112_EL2 | AMEVCNTVOFF111_EL2 | AMEVCNTVOFF110_EL2 |
| 31 | 30 | 29 | 28 | 27 | |

Bits [63:32]

Reserved, RES0.

AMEVCNTVOFF1<n>_EL2, bit [n+16], for n = 15 to 0

Indicates which implemented auxiliary counters have a corresponding virtual offset register, [AMEVCNTVOFF1<n>_EL2](#) implemented.

| AMEVCNTVOFF1<n>_EL2 | Meaning |
|---------------------|---|
| 0b0 | When read, mean that AMEVCNTR1<n>_EL0 does not have an offset, or is not implemented. |
| 0b1 | When read, means the offset AMEVCNTVOFF1<n>_EL2 is implemented for AMEVCNTR1<n>_EL0 . |

AMEVCNTR1<n>_EL0, bit [n], for n = 15 to 0

Indicates which auxiliary counters [AMEVCNTR1<n>_EL0](#) are implemented.

| AMEVCNTR1<n>_EL0 | Meaning |
|------------------|--|
| 0b0 | When read, means that AMEVCNTR1<n>_EL0 is not implemented. |
| 0b1 | When read, means that AMEVCNTR1<n>_EL0 is implemented. |

Accessing AMCG1IDR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMCG1IDR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCG1IDR_EL0;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCG1IDR_EL0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMCG1IDR_EL0;
elsif PSTATE.EL == EL3 then
    return AMCG1IDR_EL0;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCGCR_EL0, Activity Monitors Counter Group Configuration Register

The AMCGCR_EL0 characteristics are:

Purpose

Provides information on the number of activity monitor event counters implemented within each counter group.

Configuration

AArch64 System register AMCGCR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [AMCGCR\[31:0\]](#).

AArch64 System register AMCGCR_EL0 bits [31:0] are architecturally mapped to External register [AMCGCR\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCGCR_EL0 are UNDEFINED.

Attributes

AMCGCR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | CG1NC | | | | | | | | CG0NC | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

CG1NC, bits [15:8]

Counter Group 1 Number of Counters. The number of counters in the auxiliary counter group.

In an implementation that includes FEAT_AMUv1, the permitted range of values is 0x0 to 0x10.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CG0NC, bits [7:0]

Counter Group 0 Number of Counters. The number of counters in the architected counter group.

Reads as 0x04.

Access to this field is **RO**.

Accessing AMCGCR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMCGCR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCGCR_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCGCR_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCGCR_EL0;
    elsif PSTATE.EL == EL3 then
        return AMCGCR_EL0;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENCLR0_EL0, Activity Monitors Count Enable Clear Register 0

The AMCNTENCLR0_EL0 characteristics are:

Purpose

Disable control bits for the architected activity monitors event counters, [AMEVCNTR0<n>_EL0](#).

Configuration

AArch64 System register AMCNTENCLR0_EL0 bits [31:0] are architecturally mapped to AArch32 System register [AMCNTENCLR0\[31:0\]](#).

AArch64 System register AMCNTENCLR0_EL0 bits [31:0] are architecturally mapped to External register [AMCNTENCLR0\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR0_EL0 are UNDEFINED.

Attributes

AMCNTENCLR0_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | RAZ/WI | | | | | | | | | | | | P3 | P2 | P1 | P0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n>, bit [n], for n = 3 to 0

Activity monitor event counter disable bit for [AMEVCNTR0<n>_EL0](#).

Note

[AMCGCR_EL0](#).CG0NC identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

| P<n> | Meaning |
|-------------------|---|
| 0b0 | When read, means that AMEVCNTR0<n>_EL0 is disabled. When written, has no effect. |
| 0b1 | When read, means that AMEVCNTR0<n>_EL0 is enabled. When written, disables AMEVCNTR0<n>_EL0 . |

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMCNTENCLR0_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMCNTENCLR0_EL0

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b100 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMCNTEN0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCNTENCLR0_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN0 == '1'
then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCNTENCLR0_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCNTENCLR0_EL0;
    elsif PSTATE.EL == EL3 then
        return AMCNTENCLR0_EL0;

```

MSR AMCNTENCLR0_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b100 |

```

if IsHighestEL(PSTATE.EL) then
    AMCNTENCLR0_EL0 = X[t];
else
    UNDEFINED;

```


AMCNTENCLR1_EL0, Activity Monitors Count Enable Clear Register 1

The AMCNTENCLR1_EL0 characteristics are:

Purpose

Disable control bits for the auxiliary activity monitors event counters, [AMEVCNTR1<n>_EL0](#).

Configuration

AArch64 System register AMCNTENCLR1_EL0 bits [31:0] are architecturally mapped to AArch32 System register [AMCNTENCLR1\[31:0\]](#).

AArch64 System register AMCNTENCLR1_EL0 bits [31:0] are architecturally mapped to External register [AMCNTENCLR1\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR1_EL0 are UNDEFINED.

Attributes

AMCNTENCLR1_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

P<n>, bit [n], for n = 15 to 0

Activity monitor event counter disable bit for [AMEVCNTR1<n>_EL0](#).

When N is less than 16, bits [15:N] are RAZ/WI, where N is the value in [AMCGCR_EL0](#).CG1NC.

Possible values of each bit are:

| P<n> | Meaning |
|------|---|
| 0b0 | When read, means that AMEVCNTR1<n>_EL0 is disabled. When written, has no effect. |
| 0b1 | When read, means that AMEVCNTR1<n>_EL0 is enabled. When written, disables AMEVCNTR1<n>_EL0 . |

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMCNTENCLR1_EL0

If the number of auxiliary activity monitor event counters implemented is zero, reads and writes of AMCNTENCLR1_EL0 are UNDEFINED.

Note

The number of auxiliary activity monitor event counters implemented is zero exactly when [AMCFGR_EL0.NCG](#) == 0b0000.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMCNTENCLR1_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMCNTEN1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCNTENCLR1_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN1 == '1'
then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCNTENCLR1_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCNTENCLR1_EL0;
    elsif PSTATE.EL == EL3 then
        return AMCNTENCLR1_EL0;

```

MSR AMCNTENCLR1_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0011 | 0b000 |

```
if IsHighestEL(PSTATE.EL) then
    AMCNTENCLR1_EL0 = X[t];
else
    UNDEFINED;
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENSET0_EL0, Activity Monitors Count Enable Set Register 0

The AMCNTENSET0_EL0 characteristics are:

Purpose

Enable control bits for the architected activity monitors event counters, [AMEVCNTR0<n>_EL0](#).

Configuration

AArch64 System register AMCNTENSET0_EL0 bits [31:0] are architecturally mapped to AArch32 System register [AMCNTENSET0\[31:0\]](#).

AArch64 System register AMCNTENSET0_EL0 bits [31:0] are architecturally mapped to External register [AMCNTENSET0\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET0_EL0 are UNDEFINED.

Attributes

AMCNTENSET0_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | RAZ/WI | | | | | | | | | | | | P3 | P2 | P1 | P0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n>, bit [n], for n = 3 to 0

Activity monitor event counter enable bit for [AMEVCNTR0<n>_EL0](#).

Note

[AMCGCR_EL0](#).CG0NC identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

| P<n> | Meaning |
|-------------------|--|
| 0b0 | When read, means that AMEVCNTR0<n>_EL0 is disabled. When written, has no effect. |
| 0b1 | When read, means that AMEVCNTR0<n>_EL0 is enabled. When written, enables AMEVCNTR0<n>_EL0 . |

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMCNTENSET0_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMCNTENSET0_EL0

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b101 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMCNTEN0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCNTENSET0_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN0 == '1'
then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCNTENSET0_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCNTENSET0_EL0;
    elsif PSTATE.EL == EL3 then
        return AMCNTENSET0_EL0;

```

MSR AMCNTENSET0_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b101 |

```

if IsHighestEL(PSTATE.EL) then
    AMCNTENSET0_EL0 = X[t];
else
    UNDEFINED;

```


AMCNTENSET1_EL0, Activity Monitors Count Enable Set Register 1

The AMCNTENSET1_EL0 characteristics are:

Purpose

Enable control bits for the auxiliary activity monitors event counters, [AMEVCNTR1<n>_EL0](#).

Configuration

AArch64 System register AMCNTENSET1_EL0 bits [31:0] are architecturally mapped to AArch32 System register [AMCNTENSET1\[31:0\]](#).

AArch64 System register AMCNTENSET1_EL0 bits [31:0] are architecturally mapped to External register [AMCNTENSET1\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET1_EL0 are UNDEFINED.

Attributes

AMCNTENSET1_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

P<n>, bit [n], for n = 15 to 0

Activity monitor event counter enable bit for [AMEVCNTR1<n>_EL0](#).

When N is less than 16, bits [15:N] are RAZ/WI, where N is the value in [AMCGCR_EL0](#).CG1NC.

Possible values of each bit are:

| P<n> | Meaning |
|------|--|
| 0b0 | When read, means that AMEVCNTR1<n>_EL0 is disabled. When written, has no effect. |
| 0b1 | When read, means that AMEVCNTR1<n>_EL0 is enabled. When written, enables AMEVCNTR1<n>_EL0 . |

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMCNTENSET1_EL0

If the number of auxiliary activity monitor event counters implemented is zero, reads and writes of AMCNTENSET1_EL0 are UNDEFINED.

Note

The number of auxiliary activity monitor counters implemented is zero when [AMCFGR_EL0.NCG](#) == 0b0000.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMCNTENSET1_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMCNTEN1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCNTENSET1_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN1 == '1'
then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCNTENSET1_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCNTENSET1_EL0;
    elsif PSTATE.EL == EL3 then
        return AMCNTENSET1_EL0;

```

MSR AMCNTENSET1_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0011 | 0b001 |

```
if IsHighestEL(PSTATE.EL) then
    AMCNTENSET1_EL0 = X[t];
else
    UNDEFINED;
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCR_EL0, Activity Monitors Control Register

The AMCR_EL0 characteristics are:

Purpose

Global control register for the activity monitors implementation. AMCR_EL0 is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch64 System register AMCR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [AMCR\[31:0\]](#).

AArch64 System register AMCR_EL0 bits [31:0] are architecturally mapped to External register [AMCR\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCR_EL0 are UNDEFINED.

Attributes

AMCR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | CG1RZ | RES0 | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:18]

Reserved, RES0.

CG1RZ, bit [17]

When FEAT_AMUv1p1 is implemented:

Counter Group 1 Read Zero.

| CG1RZ | Meaning |
|-------|--|
| 0b0 | System register reads of AMEVCNTR1<n>_EL0 return the event count at all implemented and enabled Exception levels. |
| 0b1 | If the current Exception level is the highest implemented Exception level, system register reads of AMEVCNTR1<n>_EL0 return the event count. Otherwise, reads of AMEVCNTR1<n>_EL0 return a zero value. |

Note

Reads from the memory-mapped view are unaffected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:11]

Reserved, RES0.

HDBG, bit [10]

This bit controls whether activity monitor counting is halted when the PE is halted in Debug state.

| HDBG | Meaning |
|------|--|
| 0b0 | Activity monitors do not halt counting when the PE is halted in Debug state. |
| 0b1 | Activity monitors halt counting when the PE is halted in Debug state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [9:0]

Reserved, RES0.

Accessing AMCR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMCR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCR_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCR_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return AMCR_EL0;
    elsif PSTATE.EL == EL3 then
        return AMCR_EL0;

```

MSR AMCR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b000 |

```

if IsHighestEL(PSTATE.EL) then
    AMCR_EL0 = X[t];
else
    UNDEFINED;

```

AMEVCNTR0<n>_EL0, Activity Monitors Event Counter Registers 0, n = 0 - 3

The AMEVCNTR0<n>_EL0 characteristics are:

Purpose

Provides access to the architected activity monitor event counters.

Configuration

AArch64 System register AMEVCNTR0<n>_EL0 bits [63:0] are architecturally mapped to AArch32 System register [AMEVCNTR0<n>\[63:0\]](#).

AArch64 System register AMEVCNTR0<n>_EL0 bits [63:0] are architecturally mapped to External register [AMEVCNTR0<n>\[63:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR0<n>_EL0 are UNDEFINED.

Attributes

AMEVCNTR0<n>_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | ACNT | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ACNT | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ACNT, bits [63:0]

Architected activity monitor event counter n.

Value of architected activity monitor event counter n, where n is the number of this register and is a number from 0 to 3.

If FEAT_AMUv1p1 is implemented, [HCR_EL2](#).AMVOFFEN is 1, [SCR_EL3](#).AMVOFFEN is 1, [HCR_EL2](#).{E2H, TGE} is not {1,1}, and EL2 is implemented in the current Security state, access to these registers at EL0 or EL1 return (PCount<63:0> - [AMEVCNTVOFF0<n>_EL2](#)<63:0>).

PCount is the physical count returned when AMEVCNTR0<n>_EL0 is read from EL2 or EL3.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMEVCNTR0<n>_EL0

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVCNTR0<n>_EL0 are UNDEFINED.

Note

[AMCGCR_EL0](#).CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMEVCNTR0<n>_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|--------|
| 0b11 | 0b011 | 0b1101 | 0b010:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMEVCNTR0<n>_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVCNTR0<n>_EL0
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL3 then
    return AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)];

```

MSR AMEVCNTR0<n>_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|------------|--------|
| 0b11 | 0b011 | 0b1101 | 0b010:n[3] | n[2:0] |
|------|-------|--------|------------|--------|

```

if IsHighestEL(PSTATE.EL) then
    AMEVCNTR0_EL0[UInt(CRm<0>:op2<2:0>)] = X[t];
else
    UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVCNTR1<n>_EL0, Activity Monitors Event Counter Registers 1, n = 0 - 15

The AMEVCNTR1<n>_EL0 characteristics are:

Purpose

Provides access to the auxiliary activity monitor event counters.

Configuration

AArch64 System register AMEVCNTR1<n>_EL0 bits [63:0] are architecturally mapped to AArch32 System register [AMEVCNTR1<n>\[63:0\]](#).

AArch64 System register AMEVCNTR1<n>_EL0 bits [63:0] are architecturally mapped to External register [AMEVCNTR1<n>\[63:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR1<n>_EL0 are UNDEFINED.

Attributes

AMEVCNTR1<n>_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | ACNT | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ACNT | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ACNT, bits [63:0]

Auxiliary activity monitor event counter n.

Value of auxiliary activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If FEAT_AMUv1p1 is implemented, [HCR_EL2](#).AMVOFFEN is 1, [SCR_EL3](#).AMVOFFEN is 1, [HCR_EL2](#).{E2H, TGE} is not {1,1}, EL2 is implemented in the current Security state, and [AMCR_EL0](#).CG1RZ is 0, reads to these registers at EL0 or EL1 return (PCount<63:0> - [AMEVCNTVOFF1<n>_EL2](#)<63:0>).

PCount is the physical count returned when AMEVCNTR1<n>_EL0 is read from EL2 or EL3.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMEVCNTR1<n>_EL0

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVCNTR1<n>_EL0 are UNDEFINED.

Note

[AMCGCR_EL0](#).CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMEVCNTR1<n>_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|--------|
| 0b11 | 0b011 | 0b1101 | 0b110:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMEVCNTR1<n>_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif AMCR_EL0.CG1RZ == '1' then
            return Zeros();
        else
            return AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVCNTR1<n>_EL0
== '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsHighestEL(PSTATE.EL) && AMCR_EL0.CG1RZ == '1' then
            return Zeros();
        else
            return AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif !IsHighestEL(PSTATE.EL) && AMCR_EL0.CG1RZ == '1' then
            return Zeros();
        else
            return AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];
    elsif PSTATE.EL == EL3 then
        return AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)];

```

MSR AMEVCNTR1<n>_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|--------|
| 0b11 | 0b011 | 0b1101 | 0b110:n[3] | n[2:0] |

```

if IsHighestEL(PSTATE.EL) then
    AMEVCNTR1_EL0[UInt(CRm<0>:op2<2:0>)] = X[t];
else
    UNDEFINED;

```


AMEVCNTVOFF0<n>_EL2, Activity Monitors Event Counter Virtual Offset Registers 0, n = 0 - 15

The AMEVCNTVOFF0<n>_EL2 characteristics are:

Purpose

Holds the 64-bit virtual offset for architected activity monitor events.

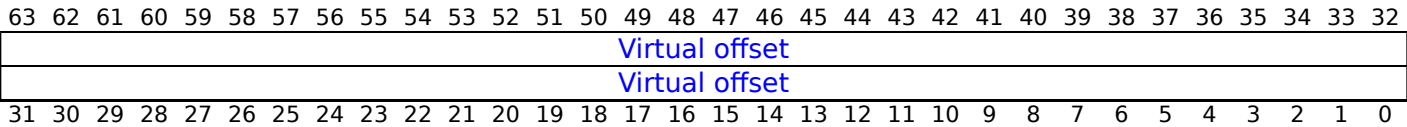
Configuration

This register is present only when FEAT_AMUv1p1 is implemented. Otherwise, direct accesses to AMEVCNTVOFF0<n>_EL2 are UNDEFINED.

Attributes

AMEVCNTVOFF0<n>_EL2 is a 64-bit register.

Field descriptions



Bits [63:0]

Virtual offset.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMEVCNTVOFF0<n>_EL2

If <n> is not 0, 2 or 3, reads and writes of AMEVCNTVOFF0<n>_EL2 are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMEVCNTVOFF0<n>_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|--------|
| 0b11 | 0b100 | 0b1101 | 0b100:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0xA00+8*UInt(CRm<0>:op2<2:0>)];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.AMV0FFEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AMV0FFEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVCNTV0FF0_EL2[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL3 then
    return AMEVCNTV0FF0_EL2[UInt(CRm<0>:op2<2:0>)];

```

MSR AMEVCNTV0FF0<n>_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|--------|
| 0b11 | 0b100 | 0b1101 | 0b100:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0xA00+8*UInt(CRm<0>:op2<2:0>)] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.AMV0FFEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AMV0FFEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMEVCNTV0FF0_EL2[UInt(CRm<0>:op2<2:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    AMEVCNTV0FF0_EL2[UInt(CRm<0>:op2<2:0>)] = X[t];

```


AMEVCNTVOFF1<n>_EL2, Activity Monitors Event Counter Virtual Offset Registers 1, n = 0 - 15

The AMEVCNTVOFF1<n>_EL2 characteristics are:

Purpose

Holds the 64-bit virtual offset for auxiliary activity monitor events.

Configuration

This register is present only when FEAT_AMUv1p1 is implemented. Otherwise, direct accesses to AMEVCNTVOFF1<n>_EL2 are UNDEFINED.

Attributes

AMEVCNTVOFF1<n>_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual offset.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMEVCNTVOFF1<n>_EL2

Note

[AMCG1IDR_ELO](#) identifies which auxiliary activity monitor event counters have a corresponding virtual offset implemented.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMEVCNTVOFF1<n>_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|--------|
| 0b11 | 0b100 | 0b1101 | 0b101:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0xA80+8*UInt(CRm<0>:op2<2:0>)];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.AMV0FFEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AMV0FFEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVCNTV0FF1_EL2[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL3 then
    return AMEVCNTV0FF1_EL2[UInt(CRm<0>:op2<2:0>)];

```

MSR AMEVCNTV0FF1<n>_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|--------|
| 0b11 | 0b100 | 0b1101 | 0b101:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0xA80+8*UInt(CRm<0>:op2<2:0>)] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.AMV0FFEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.AMV0FFEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMEVCNTV0FF1_EL2[UInt(CRm<0>:op2<2:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    AMEVCNTV0FF1_EL2[UInt(CRm<0>:op2<2:0>)] = X[t];

```


AMEVTYPER0<n>_EL0, Activity Monitors Event Type Registers 0, n = 0 - 3

The AMEVTYPER0<n>_EL0 characteristics are:

Purpose

Provides information on the events that an architected activity monitor event counter [AMEVCNTR0<n>_EL0](#) counts.

Configuration

AArch64 System register AMEVTYPER0<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register [AMEVTYPER0<n>\[31:0\]](#).

AArch64 System register AMEVTYPER0<n>_EL0 bits [31:0] are architecturally mapped to External register [AMEVTYPER0<n>\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER0<n>_EL0 are UNDEFINED.

Attributes

AMEVTYPER0<n>_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | evtCount | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the architected activity monitor event counter [AMEVCNTR0<n>_EL0](#). The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

| evtCount | Meaning | Applies when |
|----------|----------------------------|--------------|
| 0x0011 | Processor frequency cycles | When n == 0 |
| 0x4004 | Constant frequency cycles | When n == 1 |
| 0x0008 | Instructions retired | When n == 2 |
| 0x4005 | Memory stall cycles | When n == 3 |

Accessing AMEVTYPER0<n>_EL0

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVTYPER0<n>_EL0 are UNDEFINED.

Note

[AMCGCR_EL0](#).CG0NC identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMEVTYPER0<n>_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|--------|
| 0b11 | 0b011 | 0b1101 | 0b011:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVTYPER0_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVTYPER0_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMEVTYPER0_EL0[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL3 then
    return AMEVTYPER0_EL0[UInt(CRm<0>:op2<2:0>)];

```

AMEVTYPER1<n>_EL0, Activity Monitors Event Type Registers 1, n = 0 - 15

The AMEVTYPER1<n>_EL0 characteristics are:

Purpose

Provides information on the events that an auxiliary activity monitor event counter [AMEVCNTR1<n>_EL0](#) counts.

Configuration

AArch64 System register AMEVTYPER1<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register [AMEVTYPER1<n>\[31:0\]](#).

AArch64 System register AMEVTYPER1<n>_EL0 bits [31:0] are architecturally mapped to External register [AMEVTYPER1<n>\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER1<n>_EL0 are UNDEFINED.

Attributes

AMEVTYPER1<n>_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | evtCount | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the auxiliary activity monitor event counter [AMEVCNTR1<n>_EL0](#).

It is IMPLEMENTATION DEFINED what values are supported by each counter.

If software writes a value to this field which is not supported by the corresponding counter [AMEVCNTR1<n>_EL0](#), then:

- It is UNPREDICTABLE which event will be counted.
- The value read back is UNKNOWN.

The event counted by [AMEVCNTR1<n>_EL0](#) might be fixed at implementation. In this case, the field is read-only and writes are UNDEFINED.

If the corresponding counter [AMEVCNTR1<n>_EL0](#) is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMEVTYPER1<n>_EL0

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVTYPER1<n>_EL0 are UNDEFINED.

Note

[AMCGCR_EL0](#).CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMEVTYPER1<n>_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|--------|
| 0b11 | 0b011 | 0b1101 | 0b111:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HAFGRTR_EL2.AMEVTYPER1<n>_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVTYPER1<n>_EL0
== '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)];
    elsif PSTATE.EL == EL3 then
        return AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)];

```

MSR AMEVTYPER1<n>_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|--------|
| 0b11 | 0b011 | 0b1101 | 0b111:n[3] | n[2:0] |

```

if IsHighestEL(PSTATE.EL) && !boolean IMPLEMENTATION_DEFINED "AMEVCNTR1<n>_EL0 is fixed" then
    AMEVTYPER1_EL0[UInt(CRm<0>:op2<2:0>)] = X[t];
else
    UNDEFINED;

```


AMUSERENR_EL0, Activity Monitors User Enable Register

The AMUSERENR_EL0 characteristics are:

Purpose

Global user enable register for the activity monitors. Enables or disables EL0 access to the activity monitors. AMUSERENR_EL0 is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch64 System register AMUSERENR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [AMUSERENR\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMUSERENR_EL0 are UNDEFINED.

Attributes

AMUSERENR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | EN | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:1]

Reserved, RES0.

EN, bit [0]

Traps EL0 accesses to the activity monitors registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2.TGE](#) is 1, as follows:

- In AArch64 state, accesses to the following registers are trapped, reported using EC syndrome value 0x18:
 - [AMCFGR_EL0](#), [AMCGCR_EL0](#), [AMCNTENCLR0_EL0](#), [AMCNTENCLR1_EL0](#), [AMCNTENSET0_EL0](#), [AMCNTENSET1_EL0](#), [AMCR_EL0](#), [AMEVCNTR0<n>_EL0](#), [AMEVCNTR1<n>_EL0](#), [AMEVTYPER0<n>_EL0](#), and [AMEVTYPER1<n>_EL0](#).
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value 0x03, MRRC and MCRR accesses are trapped and reported using EC syndrome value 0x04:
 - [AMCFGR](#), [AMCGCR](#), [AMCNTENCLR0](#), [AMCNTENCLR1](#), [AMCNTENSET0](#), [AMCNTENSET1](#), [AMCR](#), [AMEVCNTR0<n>](#), [AMEVCNTR1<n>](#), [AMEVTYPER0<n>](#), and [AMEVTYPER1<n>](#).

| EN | Meaning |
|-----|---|
| 0b0 | EL0 accesses to the activity monitors registers are trapped. |
| 0b1 | This control does not cause any instructions to be trapped. Software can access all activity monitor registers at EL0. |

Note

- AMUSERENR_EL0 can always be read at EL0 and is not governed by this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMUSERENR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, AMUSERENR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMUSERENR_EL0;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMUSERENR_EL0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return AMUSERENR_EL0;
elsif PSTATE.EL == EL3 then
    return AMUSERENR_EL0;

```

MSR AMUSERENR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TAM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMUSERENR_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        AMUSERENR_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    AMUSERENR_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APDAKeyHi_EL1, Pointer Authentication Key A for Data (bits[127:64])

The APDAKeyHi_EL1 characteristics are:

Purpose

Holds bits[127:64] of key A used for authentication of data pointer values.

Note

The term APDAKey_EL1 is used to describe the concatenation of [APDAKeyHi_EL1](#): [APDAKeyLo_EL1](#).

Configuration

This register is present only when FEAT_PAuth is implemented. Otherwise, direct accesses to APDAKeyHi_EL1 are UNDEFINED.

Attributes

APDAKeyHi_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APDAKeyHi_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, APDAKeyHi_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.APDAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return APDAKeyHi_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APDAKeyHi_EL1;
elsif PSTATE.EL == EL3 then
    return APDAKeyHi_EL1;

```

MSR APDAKeyHi_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.APDAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDAKeyHi_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDAKeyHi_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    APDAKeyHi_EL1 = X[t];

```


APDAKeyLo_EL1, Pointer Authentication Key A for Data (bits[63:0])

The APDAKeyLo_EL1 characteristics are:

Purpose

Holds bits[63:0] of key A used for authentication of data pointer values.

Note

The term APDAKey_EL1 is used to describe the concatenation of [APDAKeyHi_EL1](#): [APDAKeyLo_EL1](#).

Configuration

This register is present only when FEAT_PAuth is implemented. Otherwise, direct accesses to APDAKeyLo_EL1 are UNDEFINED.

Attributes

APDAKeyLo_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 64 bit value, bits[63:0] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 64 bit value, bits[63:0] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APDAKeyLo_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, APDAKeyLo_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.APDAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return APDAKeyLo_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APDAKeyLo_EL1;
elsif PSTATE.EL == EL3 then
    return APDAKeyLo_EL1;

```

MSR APDAKeyLo_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.APDAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDAKeyLo_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDAKeyLo_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    APDAKeyLo_EL1 = X[t];

```


APDBKeyHi_EL1, Pointer Authentication Key B for Data (bits[127:64])

The APDBKeyHi_EL1 characteristics are:

Purpose

Holds bits[127:64] of key B used for authentication of data pointer values.

Note

The term APDBKey_EL1 is used to describe the concatenation of [APDBKeyHi_EL1](#): [APDBKeyLo_EL1](#).

Configuration

This register is present only when FEAT_PAuth is implemented. Otherwise, direct accesses to APDBKeyHi_EL1 are UNDEFINED.

Attributes

APDBKeyHi_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APDBKeyHi_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, APDBKeyHi_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.APDBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return APDBKeyHi_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APDBKeyHi_EL1;
elsif PSTATE.EL == EL3 then
    return APDBKeyHi_EL1;

```

MSR APDBKeyHi_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.APDBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDBKeyHi_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDBKeyHi_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    APDBKeyHi_EL1 = X[t];

```


APDBKeyLo_EL1, Pointer Authentication Key B for Data (bits[63:0])

The APDBKeyLo_EL1 characteristics are:

Purpose

Holds bits[63:0] of key B used for authentication of data pointer values.

Note

The term APDBKey_EL1 is used to describe the concatenation of [APDBKeyHi_EL1](#): [APDBKeyLo_EL1](#).

Configuration

This register is present only when FEAT_PAuth is implemented. Otherwise, direct accesses to APDBKeyLo_EL1 are UNDEFINED.

Attributes

APDBKeyLo_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 64 bit value, bits[63:0] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 64 bit value, bits[63:0] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APDBKeyLo_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, APDBKeyLo_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.APDBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return APDBKeyLo_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APDBKeyLo_EL1;
elsif PSTATE.EL == EL3 then
    return APDBKeyLo_EL1;

```

MSR APDBKeyLo_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.APDBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            APDBKeyLo_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APDBKeyLo_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    APDBKeyLo_EL1 = X[t];

```


APGAKeyHi_EL1, Pointer Authentication Key A for Code (bits[127:64])

The APGAKeyHi_EL1 characteristics are:

Purpose

Holds bits[127:64] of key used for generic pointer authentication code.

Note

The term APGAKey_EL1 is used to describe the concatenation of [APGAKeyHi_EL1](#): [APGAKeyLo_EL1](#).

Configuration

This register is present only when FEAT_PAuth is implemented. Otherwise, direct accesses to APGAKeyHi_EL1 are UNDEFINED.

Attributes

APGAKeyHi_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APGAKeyHi_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, APGAKeyHi_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.APGAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return APGAKeyHi_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APGAKeyHi_EL1;
elsif PSTATE.EL == EL3 then
    return APGAKeyHi_EL1;

```

MSR APGAKeyHi_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.APGAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APGAKeyHi_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APGAKeyHi_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    APGAKeyHi_EL1 = X[t];

```


APGAKeyLo_EL1, Pointer Authentication Key A for Code (bits[63:0])

The APGAKeyLo_EL1 characteristics are:

Purpose

Holds bits[63:0] of key used for generic pointer authentication code.

Note

The term APGAKey_EL1 is used to describe the concatenation of [APGAKeyHi_EL1](#): [APGAKeyLo_EL1](#).

Configuration

This register is present only when FEAT_PAuth is implemented. Otherwise, direct accesses to APGAKeyLo_EL1 are UNDEFINED.

Attributes

APGAKeyLo_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 64 bit value, bits[63:0] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 64 bit value, bits[63:0] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APGAKeyLo_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, APGAKeyLo_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.APGAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return APGAKeyLo_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APGAKeyLo_EL1;
elsif PSTATE.EL == EL3 then
    return APGAKeyLo_EL1;

```

MSR APGAKeyLo_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.APGAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APGAKeyLo_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APGAKeyLo_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    APGAKeyLo_EL1 = X[t];

```


APIAKeyHi_EL1, Pointer Authentication Key A for Instruction (bits[127:64])

The APIAKeyHi_EL1 characteristics are:

Purpose

Holds bits[127:64] of key A used for authentication of instruction pointer values.

Note

The term APIAKey_EL1 is used to describe the concatenation of [APIAKeyHi_EL1](#): [APIAKeyLo_EL1](#).

Configuration

This register is present only when FEAT_PAuth is implemented. Otherwise, direct accesses to APIAKeyHi_EL1 are UNDEFINED.

Attributes

APIAKeyHi_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APIAKeyHi_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, APIAKeyHi_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.APIAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return APIAKeyHi_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APIAKeyHi_EL1;
elsif PSTATE.EL == EL3 then
    return APIAKeyHi_EL1;

```

MSR APIAKeyHi_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.APIAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIAKeyHi_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIAKeyHi_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    APIAKeyHi_EL1 = X[t];

```


APIAKeyLo_EL1, Pointer Authentication Key A for Instruction (bits[63:0])

The APIAKeyLo_EL1 characteristics are:

Purpose

Holds bits[63:0] of key A used for authentication of instruction pointer values.

Note

The term APIAKey_EL1 is used to describe the concatenation of [APIAKeyHi_EL1](#): [APIAKeyLo_EL1](#).

Configuration

This register is present only when FEAT_PAuth is implemented. Otherwise, direct accesses to APIAKeyLo_EL1 are UNDEFINED.

Attributes

APIAKeyLo_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 64 bit value, bits[63:0] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 64 bit value, bits[63:0] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APIAKeyLo_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, APIAKeyLo_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.APIAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return APIAKeyLo_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APIAKeyLo_EL1;
elsif PSTATE.EL == EL3 then
    return APIAKeyLo_EL1;

```

MSR APIAKeyLo_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.APIAKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIAKeyLo_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIAKeyLo_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    APIAKeyLo_EL1 = X[t];

```


APIBKeyHi_EL1, Pointer Authentication Key B for Instruction (bits[127:64])

The APIBKeyHi_EL1 characteristics are:

Purpose

Holds bits[127:64] of key B used for authentication of instruction pointer values.

Note

The term APIBKey_EL1 is used to describe the concatenation of [APIBKeyHi_EL1](#): [APIBKeyLo_EL1](#).

Configuration

This register is present only when FEAT_PAuth is implemented. Otherwise, direct accesses to APIBKeyHi_EL1 are UNDEFINED.

Attributes

APIBKeyHi_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 64 bit value, bits[127:64] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

64 bit value, bits[127:64] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APIBKeyHi_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, APIBKeyHi_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0001 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.APIBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return APIBKeyHi_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APIBKeyHi_EL1;
elsif PSTATE.EL == EL3 then
    return APIBKeyHi_EL1;

```

MSR APIBKeyHi_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0001 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.APIBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIBKeyHi_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIBKeyHi_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    APIBKeyHi_EL1 = X[t];

```


APIBKeyLo_EL1, Pointer Authentication Key B for Instruction (bits[63:0])

The APIBKeyLo_EL1 characteristics are:

Purpose

Holds bits[63:0] of key B used for authentication of instruction pointer values.

Note

The term APIBKey_EL1 is used to describe the concatenation of [APIBKeyHi_EL1](#): [APIBKeyLo_EL1](#).

Configuration

This register is present only when FEAT_PAuth is implemented. Otherwise, direct accesses to APIBKeyLo_EL1 are UNDEFINED.

Attributes

APIBKeyLo_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| 64 bit value, bits[63:0] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 64 bit value, bits[63:0] of the 128 bit pointer authentication key value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

64 bit value, bits[63:0] of the 128 bit pointer authentication key value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APIBKeyLo_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, APIBKeyLo_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.APIBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return APIBKeyLo_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return APIBKeyLo_EL1;
elsif PSTATE.EL == EL3 then
    return APIBKeyLo_EL1;

```

MSR APIBKeyLo_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.APK == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.APIBKey == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            APIBKeyLo_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.APK == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.APK == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        APIBKeyLo_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    APIBKeyLo_EL1 = X[t];

```


AT S12E0R, Address Translate Stages 1 and 2 EL0 Read

The AT S12E0R characteristics are:

Purpose

Performs stage 1 and 2 address translations from EL0, with permissions as if reading from the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of [SCR_EL3.NS](#):
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the EL1&0 translation regime.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

Configuration

There are no configuration notes.

Attributes

AT S12E0R is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S12E0R instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S12E0R, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b0111 | 0b1000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00' then
        AArch64.AT(X[t], TranslationStage_1, EL0, ATAccess_Read);
    else
        AArch64.AT(X[t], TranslationStage_12, EL0, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.AT(X[t], TranslationStage_1, EL0, ATAccess_Read);
    elsif EL2Enabled() && (HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00') then
        AArch64.AT(X[t], TranslationStage_1, EL0, ATAccess_Read);
    else
        AArch64.AT(X[t], TranslationStage_12, EL0, ATAccess_Read);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S12E0W, Address Translate Stages 1 and 2 EL0 Write

The AT S12E0W characteristics are:

Purpose

Performs stage 1 and 2 address translations from EL0, with permissions as if writing to the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of [SCR_EL3.NS](#):
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the EL1&0 translation regime.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

Configuration

There are no configuration notes.

Attributes

AT S12E0W is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S12E0W instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S12E0W, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b0111 | 0b1000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00' then
        AArch64.AT(X[t], TranslationStage_1, EL0, ATAccess_Write);
    else
        AArch64.AT(X[t], TranslationStage_12, EL0, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.AT(X[t], TranslationStage_1, EL0, ATAccess_Write);
    elsif EL2Enabled() && (HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00') then
        AArch64.AT(X[t], TranslationStage_1, EL0, ATAccess_Write);
    else
        AArch64.AT(X[t], TranslationStage_12, EL0, ATAccess_Write);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S12E1R, Address Translate Stages 1 and 2 EL1 Read

The AT S12E1R characteristics are:

Purpose

Performs stage 1 and 2 address translation, with permissions as if reading from the given virtual address from EL1, or from EL2 if the Effective value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of [SCR_EL3](#).NS:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

Configuration

There are no configuration notes.

Attributes

AT S12E1R is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S12E1R instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S12E1R, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b0111 | 0b1000 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00' then
        AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_Read);
    else
        AArch64.AT(X[t], TranslationStage_12, EL1, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_Read);
    elsif EL2Enabled() && (HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00') then
        AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_Read);
    else
        AArch64.AT(X[t], TranslationStage_12, EL1, ATAccess_Read);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S12E1W, Address Translate Stages 1 and 2 EL1 Write

The AT S12E1W characteristics are:

Purpose

Performs stage 1 and 2 address translation, with permissions as if writing to the given virtual address from EL1, or from EL2 if the Effective value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of [SCR_EL3](#).NS:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

Configuration

There are no configuration notes.

Attributes

AT S12E1W is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S12E1W instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S12E1W, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b0111 | 0b1000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00' then
        AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_Write);
    else
        AArch64.AT(X[t], TranslationStage_12, EL1, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_Write);
    elsif EL2Enabled() && (HCR_EL2.<E2H,TGE> == '11' || HCR_EL2.<DC,VM> == '00') then
        AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_Write);
    else
        AArch64.AT(X[t], TranslationStage_12, EL1, ATAccess_Write);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E0R, Address Translate Stage 1 EL0 Read

The AT S1E0R characteristics are:

Purpose

Performs stage 1 address translation from EL0, with permissions as if reading from the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of [SCR_EL3.NS](#):
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the EL1&0 translation regime.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

Configuration

There are no configuration notes.

Attributes

AT S1E0R is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E0R instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S1E0R, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b1000 | 0b010 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.ATS1E0R == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.AT(X[t], TranslationStage_1, EL0, ATAccess_Read);
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t], TranslationStage_1, EL0, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t], TranslationStage_1, EL0, ATAccess_Read);
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E0W, Address Translate Stage 1 EL0 Write

The AT S1E0W characteristics are:

Purpose

Performs stage 1 address translation from EL0, with permissions as if writing to the given virtual address from EL0, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of [SCR_EL3.NS](#):
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the EL1&0 translation regime.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the EL2&0 translation regime.
- Otherwise, the EL1&0 translation regime.

Configuration

There are no configuration notes.

Attributes

AT S1E0W is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E0W instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S1E0W, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b1000 | 0b011 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.ATS1E0W == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.AT(X[t], TranslationStage_1, EL0, ATAccess_Write);
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t], TranslationStage_1, EL0, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t], TranslationStage_1, EL0, ATAccess_Write);
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E1R, Address Translate Stage 1 EL1 Read

The AT S1E1R characteristics are:

Purpose

Performs stage 1 address translation, with permissions as if reading from the given virtual address from EL1, or from EL2 if the Effective value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of [SCR_EL3](#).NS:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

Configuration

There are no configuration notes.

Attributes

AT S1E1R is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E1R instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S1E1R, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b1000 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.ATS1E1R == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_Read);
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_Read);
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E1RP, Address Translate Stage 1 EL1 Read PAN

The AT S1E1RP characteristics are:

Purpose

Performs a stage 1 address translation, where the value of PSTATE.PAN determines if a read from a location will generate a Permission fault for a privileged access, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of [SCR_EL3.NS](#):
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

Configuration

This instruction is present only when FEAT_PAN2 is implemented. Otherwise, direct accesses to AT S1E1RP are UNDEFINED.

Attributes

AT S1E1RP is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E1RP instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S1E1RP, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b1001 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.ATS1E1RP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_ReadPAN);
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_ReadPAN);
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_ReadPAN);
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E1W, Address Translate Stage 1 EL1 Write

The AT S1E1W characteristics are:

Purpose

Performs stage 1 address translation, with permissions as if writing to the given virtual address from EL1, or from EL2 if the Effective value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of [SCR_EL3](#).NS:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

Configuration

There are no configuration notes.

Attributes

AT S1E1W is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E1W instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S1E1W, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b1000 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.ATS1E1W == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_Write);
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_Write);
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E1WP, Address Translate Stage 1 EL1 Write PAN

The AT S1E1WP characteristics are:

Purpose

Performs a stage 1 address translation, where the value of PSTATE.PAN determines if a write to a location will generate a Permission fault for a privileged access, using the following translation regime:

- When EL2 is implemented and enabled in the Security state described by the current value of [SCR_EL3.NS](#):
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the EL1&0 translation regime, accessed from EL1.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the EL2&0 translation regime, accessed from EL2.
- Otherwise, the EL1&0 translation regime, accessed from EL1.

Configuration

This instruction is present only when FEAT_PAN2 is implemented. Otherwise, direct accesses to AT S1E1WP are UNDEFINED.

Attributes

AT S1E1WP is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E1WP instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S1E1WP, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b1001 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.ATS1E1WP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_WritePAN);
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_WritePAN);
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t], TranslationStage_1, EL1, ATAccess_WritePAN);
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AT S1E2R, Address Translate Stage 1 EL2 Read

The AT S1E2R characteristics are:

Purpose

Performs stage 1 address translation as defined for EL2, with permissions as if reading from the given virtual address.

Configuration

There are no configuration notes.

Attributes

AT S1E2R is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E2R instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S1E2R, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b0111 | 0b1000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t], TranslationStage_1, EL2, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        AArch64.AT(X[t], TranslationStage_1, EL2, ATAccess_Read);

```


AT S1E2W, Address Translate Stage 1 EL2 Write

The AT S1E2W characteristics are:

Purpose

Performs stage 1 address translation as defined for EL2, with permissions as if writing to the given virtual address.

Configuration

There are no configuration notes.

Attributes

AT S1E2W is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E2W instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S1E2W, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b0111 | 0b1000 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.AT(X[t], TranslationStage_1, EL2, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        AArch64.AT(X[t], TranslationStage_1, EL2, ATAccess_Write);
```


AT S1E3R, Address Translate Stage 1 EL3 Read

The AT S1E3R characteristics are:

Purpose

Performs stage 1 address translation as defined for EL3, with permissions as if reading from the given virtual address.

Configuration

There are no configuration notes.

Attributes

AT S1E3R is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E3R instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S1E3R, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b0111 | 0b1000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t], TranslationStage_1, EL3, ATAccess_Read);

```

AT S1E3W, Address Translate Stage 1 EL3 Write

The AT S1E3W characteristics are:

Purpose

Performs stage 1 address translation as defined for EL3, with permissions as if writing to the given virtual address.

Configuration

There are no configuration notes.

Attributes

AT S1E3W is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:0]

Input address for translation. The resulting address can be read from the [PAR_EL1](#).

If the address translation instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then VA[63:32] is RES0.

Executing the AT S1E3W instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

AT S1E3W, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b0111 | 0b1000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.AT(X[t], TranslationStage_1, EL3, ATAccess_Write);

```

BRB IALL, Invalidate the Branch Record Buffer

The BRB IALL characteristics are:

Purpose

Invalidates all Branch records in the Branch Record Buffer.

Configuration

This instruction is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRB IALL are UNDEFINED.

Attributes

BRB IALL is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the BRB IALL instruction

Rt should be encoded as 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

BRB IALL

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b001 | 0b0111 | 0b0010 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.nBRBIALL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRB_IALL();
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRB_IALL();
elsif PSTATE.EL == EL3 then
    BRB_IALL();

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRB INJ, Branch Record Injection into the Branch Record Buffer

The BRB INJ characteristics are:

Purpose

Injects the Branch Record held in [BRBINFINJ_EL1](#), [BRBSRCINJ_EL1](#), and [BRBTGTINJ_EL1](#) into the Branch Record Buffer.

Configuration

This instruction is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRB INJ are UNDEFINED.

Attributes

BRB INJ is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the BRB INJ instruction

Rt should be encoded as 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

BRB INJ

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b001 | 0b0111 | 0b0010 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.nBRBINJ == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRB_INJ();
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRB_INJ();
elsif PSTATE.EL == EL3 then
    BRB_INJ();

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBCR_EL1, Branch Record Buffer Control Register (EL1)

The BRBCR_EL1 characteristics are:

Purpose

Controls the Branch Record Buffer.

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBCR_EL1 are UNDEFINED.

Attributes

BRBCR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-----------|----|----|----|------|----|----|----|------|----|----|----|----|----|----|----|-----|----|------|----|----|----|-------|----|----|----|------|----|-------|--|-------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | EXCEPTION | | | | ERTN | | | | RES0 | | | | | | | | FZP | | RES0 | | TS | | MPRED | | CC | | RES0 | | E1BRE | | E0BRE | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

Bits [63:24]

Reserved, RES0.

EXCEPTION, bit [23]

Enable the recording of entry to EL1 via an exception.

| EXCEPTION | Meaning |
|-----------|---|
| 0b0 | Disable the recording of Branch records for exceptions when taken to EL1. |
| 0b1 | Enable the recording of Branch records for exceptions when taken to EL1. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

ERTN, bit [22]

Allow the recording Branch records for exception return instructions from EL1.

| ERTN | Meaning |
|------|--|
| 0b0 | Disable the recording Branch records for exception return instructions from EL1. |
| 0b1 | Enable the recording Branch records for exception return instructions from EL1. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bits [21:9]

Reserved, RES0.

FZP, bit [8]

When FEAT_PMUv3 is implemented:

Freeze BRBE on PMU overflow.

| FZP | Meaning |
|-----|--|
| 0b0 | Branch recording is not affected by this control. |
| 0b1 | A BRBE freeze event occurs when a PMU overflow occurs. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control.

| TS | Meaning | Applies when |
|------|--|------------------------------|
| 0b01 | Virtual timestamp. The BRBE recorded timestamp is the physical counter value, minus the value of CNTVOFF_EL2 . | When FEAT_ECV is implemented |
| 0b10 | Guest physical timestamp. The BRBE recorded timestamp is the physical counter value minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2 : <ul style="list-style-type: none"> • EL3 is implemented and SCR_EL3.ECVEn == 0. • EL2 is implemented and CNTHCTL_EL2.ECV == 0. | |
| 0b11 | Physical timestamp. The BRBE recorded timestamp is the physical counter value. | |

All other values are reserved.

This field is ignored by the PE when EL2 is implemented and [BRBCR_EL2](#).TS != 0b00.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.

- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

MPRED, bit [4]

Mask the recording of mispredicts.

| MPRED | Meaning |
|-------|--|
| 0b0 | Disable the recording of mispredict information. |
| 0b1 | Allow the recording of mispredict information. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

CC, bit [3]

Enable the recording of cycle count information.

| CC | Meaning |
|-----|---|
| 0b0 | Disable the recording of cycle count information. |
| 0b1 | Allow the recording of cycle count information. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E1BRE, bit [1]

EL1 Branch recording enable.

| E1BRE | Meaning |
|-------|-------------------------------------|
| 0b0 | Branch recording prohibited at EL1. |
| 0b1 | Branch recording enabled at EL1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

E0BRE, bit [0]

EL0 Branch recording enable.

| E0BRE | Meaning |
|-------|-------------------------------------|
| 0b0 | Branch recording prohibited at EL0. |
| 0b1 | Branch recording enabled at EL0. |

This field is ignored by the PE when EL2 is implemented and enabled in the current Security state and [HCR_EL2.TGE](#) == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing BRBCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, BRBCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            return NVMem[0x8E0];
        else
            return BRBCR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            return BRBCR_EL2;
        else
            return BRBCR_EL1;
    elsif PSTATE.EL == EL3 then
        return BRBCR_EL1;

```

MRS <Xt>, BRBCR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b101 | 0b1001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x8E0];
    elseif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            UNDEFINED;
        elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3
trap priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
            UNDEFINED;
        elseif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elseif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return BRBCR_EL1;
    else
        UNDEFINED;
elseif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return BRBCR_EL1;
    else
        UNDEFINED;

```

MSR BRBCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x8E0] = X[t];
    else
        BRBCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        BRBCR_EL2 = X[t];
    else
        BRBCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    BRBCR_EL1 = X[t];

```

MSR BRBCR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b101 | 0b1001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x8E0] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3
trap priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            BRBCR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        BRBCR_EL1 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBCR_EL2, Branch Record Buffer Control Register (EL2)

The BRBCR_EL2 characteristics are:

Purpose

Controls the Branch Record Buffer.

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBCR_EL2 are UNDEFINED.

Attributes

BRBCR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | EXCEPTION | | | | | | | | ERTN | | | | | | | | RES0 | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

EXCEPTION, bit [23]

Enable the recording of entry to EL2 via an exception.

| EXCEPTION | Meaning |
|-----------|---|
| 0b0 | Disable the recording of Branch records for exceptions when taken to EL2. |
| 0b1 | Enable the recording of Branch records for exceptions when taken to EL2. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

ERTN, bit [22]

Allow the recording Branch records for exception return instructions from EL2.

| ERTN | Meaning |
|------|--|
| 0b0 | Disable the recording Branch records for exception return instructions from EL2. |
| 0b1 | Enable the recording Branch records for exception return instructions from EL2. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bits [21:9]

Reserved, RES0.

FZP, bit [8]

When FEAT_PMUv3 is implemented:

Freeze BRBE on PMU overflow.

| FZP | Meaning |
|-----|--|
| 0b0 | Branch recording is not affected by this control. |
| 0b1 | A BRBE freeze event occurs when a PMU overflow occurs. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control.

| TS | Meaning | Applies when |
|------|--|------------------------------|
| 0b00 | Timestamp controlled by BRBCR_EL1.TS . | |
| 0b01 | Virtual timestamp. The BRBE recorded timestamp is the physical counter value, minus the value of CNTVOFF_EL2 . | |
| 0b10 | Guest physical timestamp. The BRBE recorded timestamp is the physical counter value minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2 : <ul style="list-style-type: none"> • EL3 is implemented and SCR_EL3.ECVEn == 0. • EL2 is implemented and CNTHCTL_EL2.ECV == 0. | When FEAT_ECV is implemented |
| 0b11 | Physical timestamp. The BRBE recorded timestamp is the physical counter value. | |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

MPRED, bit [4]

Mask the recording of mispredicts.

| MPRED | Meaning |
|--------------|--|
| 0b0 | Disable the recording of mispredict information. |
| 0b1 | Allow the recording of mispredict information. |

If EL2 is not implemented, then the Effective value of this field is 1, other than for a direct read of the register.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

CC, bit [3]

Enable the recording of cycle count information.

| CC | Meaning |
|-----------|---|
| 0b0 | Disable the recording of cycle count information. |
| 0b1 | Allow the recording of cycle count information. |

If EL2 is not implemented, then the Effective value of this field is 1, other than for a direct read of the register.

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E2BRE, bit [1]

EL2 Branch recording enable.

| E2BRE | Meaning |
|--------------|-------------------------------------|
| 0b0 | Branch recording prohibited at EL2. |
| 0b1 | Branch recording enabled at EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

EOHBRE, bit [0]

EL0 Branch recording enable.

| EOHBRE | Meaning |
|---------------|---|
| 0b0 | Branch recording prohibited at EL0 when HCR_EL2.TGE == 1. |
| 0b1 | Branch recording enabled at EL0 when HCR_EL2.TGE == 1. |

This field is ignored by the PE when any of the following are true:

- [HCR_EL2.TGE](#) == 0.
- EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing BRBCR_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, accesses from EL2 using the register name BRBCR_EL2 or BRBCR_EL1 are not guaranteed to be ordered with respect to accesses using the other register name.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, BRBCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x8E0];
    else
        return BRBCR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return BRBCR_EL2;
    else
        return BRBCR_EL1;
elsif PSTATE.EL == EL3 then
    return BRBCR_EL1;

```

MRS <Xt>, BRBCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b100 | 0b1001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBCR_EL2;
elsif PSTATE.EL == EL3 then
    return BRBCR_EL2;

```

MSR BRBCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x8E0] = X[t];
    else
        BRBCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        BRBCR_EL2 = X[t];
    else
        BRBCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    BRBCR_EL1 = X[t];

```

MSR BRBCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b100 | 0b1001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elseif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBCR_EL2 = X[t];
elseif PSTATE.EL == EL3 then
    BRBCR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBFCCR_EL1, Branch Record Buffer Function Control Register

The BRBFCCR_EL1 characteristics are:

Purpose

Functional controls for the Branch Record Buffer.

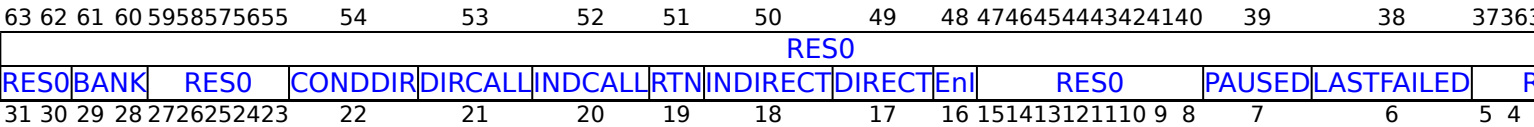
Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBFCCR_EL1 are UNDEFINED.

Attributes

BRBFCCR_EL1 is a 64-bit register.

Field descriptions



Bits [63:30]

Reserved, RES0.

BANK, bits [29:28]

Branch record buffer bank access control.

| BANK | Meaning |
|------|---------------------------------|
| 0b00 | Select branch records 0 to 31. |
| 0b01 | Select branch records 32 to 63. |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:23]

Reserved, RES0.

CONDDIR, bit [22]

Match on conditional direct branch instructions.

| CONDDIR | Meaning |
|---------|---|
| 0b0 | Do not match on conditional direct branch instructions. |
| 0b1 | Match on conditional direct branch instructions. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

DIRCALL, bit [21]

Match on direct branch with link instructions.

| DIRCALL | Meaning |
|----------------|---|
| 0b0 | Do not match on direct branch with link instructions. |
| 0b1 | Match on direct branch with link instructions. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

INDCALL, bit [20]

Match on indirect branch with link instructions.

| INDCALL | Meaning |
|----------------|---|
| 0b0 | Do not match on indirect branch with link instructions. |
| 0b1 | Match on indirect branch with link instructions. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

RTN, bit [19]

Match on function return instructions.

| RTN | Meaning |
|------------|---|
| 0b0 | Do not match on function return instructions. |
| 0b1 | Match on function return instructions. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

INDIRECT, bit [18]

Match on indirect branch instructions.

| INDIRECT | Meaning |
|-----------------|---|
| 0b0 | Do not match on indirect branch instructions. |
| 0b1 | Match on indirect branch instructions. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

DIRECT, bit [17]

Match on unconditional direct branch instructions.

| DIRECT | Meaning |
|---------------|---|
| 0b0 | Do not match on unconditional direct branch instructions. |
| 0b1 | Match on unconditional direct branch instructions. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

EnI, bit [16]

Include or exclude matches.

| EnI | Meaning |
|------------|---|
| 0b0 | Include records for matches, and exclude records for non-matches. |
| 0b1 | Exclude records for matches, and include records for non-matches. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Bits [15:8]

Reserved, RES0.

PAUSED, bit [7]

Branch recording Paused status.

| PAUSED | Meaning |
|---------------|---------------------------------|
| 0b0 | Branch recording is not Paused. |
| 0b1 | Branch recording is Paused. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

LASTFAILED, bit [6]

When FEAT_TME is implemented:

Indicates transaction failure or cancellation.

| LASTFAILED | Meaning |
|-------------------|--|
| 0b0 | Indicates that no transactions in a non-prohibited region have failed or been canceled since the last Branch record was generated. |
| 0b1 | Indicates that at least one transaction in a non-prohibited region has failed or been canceled since the last Branch record was generated. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_BRBEv1p1 is implemented, this field resets to an architecturally UNKNOWN value.
- On a Warm reset, when FEAT_BRBEv1p1 is not implemented, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [5:0]

Reserved, RES0.

Accessing BRBFCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, BRBFCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBFCR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBFCR_EL1;
elsif PSTATE.EL == EL3 then
    return BRBFCR_EL1;

```

MSR BRBFCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.nBRBCTL == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBFCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBFCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    BRBFCR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBIDR0_EL1, Branch Record Buffer ID0 Register

The BRBIDR0_EL1 characteristics are:

Purpose

Indicates the features of the branch buffer unit.

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBIDR0_EL1 are UNDEFINED.

Attributes

BRBIDR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|--------|----|----|----|--------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | RES0 | | | | | | | | RES0 | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | CC | | | | FORMAT | | | | NUMREC | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

CC, bits [15:12]

Cycle counter support. Defined values are:

| CC | Meaning |
|--------|-----------------------------------|
| 0b0101 | 20-bit cycle counter implemented. |

All other values are reserved.

FORMAT, bits [11:8]

Data format of records of the Branch record buffer. Defined values are:

| FORMAT | Meaning |
|--------|-----------|
| 0b0000 | Format 0. |

All other values are reserved.

NUMREC, bits [7:0]

Number of records supported. Defined values are:

| NUMREC | Meaning |
|--------|--------------------------------|
| 0x08 | 8 branch records implemented. |
| 0x10 | 16 branch records implemented. |
| 0x20 | 32 branch records implemented. |
| 0x40 | 64 branch records implemented. |

All other values are reserved.

Accessing BRBIDR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, BRBIDR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.nBRBIDR == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBIDR0_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBIDR0_EL1;
elsif PSTATE.EL == EL3 then
    return BRBIDR0_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBINFINJ_EL1, Branch Record Buffer Information Injection Register

The BRBINFINJ_EL1 characteristics are:

Purpose

The information of a Branch record for injection.

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBINFINJ_EL1 are UNDEFINED.

Attributes

BRBINFINJ_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|------|----|------|----|----|----|----|----|----|----|-------|----|------|----|-------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | CCU | | CC | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | LASTFAILED | | T | RES0 | | TYPE | | | | | | EL | | MPRED | | RES0 | | VALID | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:47]

Reserved, RES0.

CCU, bit [46]

The number of PE clock cycles since the last Branch record entry is UNKNOWN.

| CCU | Meaning |
|-----|---|
| 0b0 | Indicates that the number of PE clock cycles since the last Branch record is indicated by BRBINFINJ_EL1.CC. |
| 0b1 | Indicates that the number of PE clock cycles since the last Branch record is UNKNOWN. |

The value in this field is only valid when BRBINFINJ_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINFINJ_EL1.VALID == 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

CC, bits [45:32]

The number of PE clock cycles since the last Branch record entry.

The format of this field uses a mantissa and exponent to express the cycle count value, as follows:

- CC bits[7:0] indicate the mantissa M.

- CC bits[13:8] indicate the exponent E.

The cycle count is expressed using the following function:

if IsZero(E) then UInt(M) else UInt('1':M:Zeros(UInt(E)-1))

If required, the cycle count is rounded to a multiple of $2^{(E-1)}$ towards zero before being encoded.

A value of all ones in both the mantissa and exponent indicates the cycle count value exceeded the size of the cycle counter.

The value in this field is only valid when BRBINFINJ_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- **RES0** if any of the following are true:
 - BRBINFINJ_EL1.CCU == 1
 - BRBINFINJ_EL1.VALID == 0b00
- Otherwise, access to this field is **RW**.

Bits [31:18]

Reserved, RES0.

LASTFAILED, bit [17]

When FEAT_TME is implemented:

Indicates transaction failure or cancellation.

| LASTFAILED | Meaning |
|------------|---|
| 0b0 | Indicates that no transactions in a non-prohibited region have failed or been canceled between the previous Branch record and this Branch record. |
| 0b1 | Indicates that at least one transaction in a non-prohibited region has failed or been canceled between the previous Branch record and this Branch record. |

The value in this field is only valid when BRBINFINJ_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINFINJ_EL1.VALID == 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

T, bit [16]

When FEAT_TME is implemented:

Transactional state.

| T | Meaning |
|-----|--|
| 0b0 | The branch or exception was not executed in Transactional state. |
| 0b1 | The branch or exception was executed in Transactional state. |

The value in this field is only valid when BRBINFINJ_EL1.VALID == 0b10 or BRBINFINJ_EL1.VALID == 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- RES0** if any of the following are true:
 - BRBINFINJ_EL1.VALID == 0b00
 - BRBINFINJ_EL1.VALID == 0b01
- Otherwise, access to this field is **RW**.

Otherwise:

Reserved, RES0.

Bits [15:14]

Reserved, RES0.

TYPE, bits [13:8]

Branch type.

| TYPE | Meaning |
|----------|--|
| 0b000000 | Unconditional direct branch, excluding Branch with link. |
| 0b000001 | Indirect branch, excluding Branch with link, Return from subroutine, and Exception return. |
| 0b000010 | Direct Branch with link. |
| 0b000011 | Indirect Branch with link. |
| 0b000101 | Return from subroutine. |
| 0b000111 | Exception return. |
| 0b001000 | Conditional direct branch. |
| 0b100001 | Debug halt. |
| 0b100010 | Call. |
| 0b100011 | Trap. |
| 0b100100 | SError. |
| 0b100110 | Instruction debug. |
| 0b100111 | Data debug. |
| 0b101010 | Alignment. |
| 0b101011 | Inst Fault. |
| 0b101100 | Data Fault. |
| 0b101110 | IRQ. |
| 0b101111 | FIQ. |
| 0b111001 | Debug State Exit. |

All other values are reserved.

The value in this field is only valid when BRBINFINJ_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINFINJ_EL1.VALID == 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

EL, bits [7:6]

The Exception Level at the target address.

| EL | Meaning | Applies when |
|------|---------|-----------------------------------|
| 0b00 | EL0. | |
| 0b01 | EL1. | |
| 0b10 | EL2. | |
| 0b11 | EL3. | When FEAT_BRBEv1p1 is implemented |

The value in this field is only valid when BRBINFINJ_EL1.VALID == 0b11 or BRBINFINJ_EL1.VALID == 0b01.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- **RES0** if any of the following are true:
 - BRBINFINJ_EL1.VALID == 0b00
 - BRBINFINJ_EL1.VALID == 0b10
- Otherwise, access to this field is **RW**.

MPRED, bit [5]

Branch mispredict.

| MPRED | Meaning |
|-------|--|
| 0b0 | Branch was correctly predicted or the result of the prediction was not captured. |
| 0b1 | Branch was incorrectly predicted. |

The value in this field is only valid when BRBINFINJ_EL1.VALID == 0b11 or BRBINFINJ_EL1.VALID == 0b10.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- **RES0** if any of the following are true:
 - BRBINFINJ_EL1.VALID == 0b00
 - BRBINFINJ_EL1.VALID == 0b01
 - BRBINFINJ_EL1.TYPE[5] == 1
- Otherwise, access to this field is **RW**.

Bits [4:2]

Reserved, RES0.

VALID, bits [1:0]

The Branch record is valid.

| VALID | Meaning |
|-------|--|
| 0b00 | This Branch record is not valid. The values of following fields are not valid: <ul style="list-style-type: none"> • BRBTGTINJ_EL1.ADDRESS. • BRBSRCINJ_EL1.ADDRESS. • BRBINFINJ_EL1.LASTFAILED. • BRBINFINJ_EL1.T. • BRBINFINJ_EL1.EL. • BRBINFINJ_EL1.TYPE. • BRBINFINJ_EL1.CC. • BRBINFINJ_EL1.CCU. |
| 0b01 | This Branch record is valid. The values of following fields are not valid: <ul style="list-style-type: none"> • BRBSRCINJ_EL1.ADDRESS. • BRBINFINJ_EL1.T. • BRBINFINJ_EL1.MPRED. |
| 0b10 | This Branch record is valid. The values of following fields are not valid: <ul style="list-style-type: none"> • BRBTGTINJ_EL1.ADDRESS. • BRBINFINJ_EL1.EL. |
| 0b11 | This Branch record is valid. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing BRBINFINJ_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, BRBINFINJ_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.nBRBDATA == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBINFINJ_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBINFINJ_EL1;
elsif PSTATE.EL == EL3 then
    return BRBINFINJ_EL1;

```

MSR BRBINFINJ_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.nBRBDATA == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBINFINJ_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBINFINJ_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    BRBINFINJ_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBINF<n>_EL1, Branch Record Buffer Information Register <n>, n = 0 - 31

The BRBINF<n>_EL1 characteristics are:

Purpose

The information for Branch record n + ([BRBFCCR_EL1](#).BANK × 32).

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBINF<n>_EL1 are UNDEFINED.

Attributes

BRBINF<n>_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|------|----|------|----|----|----|----|----|----|-------|------|----|-------|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | CCU | | CC | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | LASTFAILED | | T | RES0 | | TYPE | | | | | | EL | MPRED | RES0 | | VALID | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:47]

Reserved, RES0.

CCU, bit [46]

The number of PE clock cycles since the last Branch record entry is UNKNOWN.

| CCU | Meaning |
|-----|---|
| 0b0 | Indicates that the number of PE clock cycles since the last Branch record is indicated by BRBINF<n>_EL1.CC. |
| 0b1 | Indicates that the number of PE clock cycles since the last Branch record is UNKNOWN. |

The value in this field is only valid when BRBINF<n>_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINF<n>_EL1.VALID == 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RO**.

CC, bits [45:32]

The number of PE clock cycles since the last Branch record entry.

The format of this field uses a mantissa and exponent to express the cycle count value, as follows:

- CC bits[7:0] indicate the mantissa M.

- CC bits[13:8] indicate the exponent E.

The cycle count is expressed using the following function:

if IsZero(E) then UInt(M) else UInt('1':M:Zeros(UInt(E)-1))

If required, the cycle count is rounded to a multiple of $2^{(E-1)}$ towards zero before being encoded.

A value of all ones in both the mantissa and exponent indicates the cycle count value exceeded the size of the cycle counter.

The value in this field is only valid when BRBINF<n>_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- **RES0** if any of the following are true:
 - BRBINF<n>_EL1.CCU == 1
 - BRBINF<n>_EL1.VALID == 0b00
- Otherwise, access to this field is **RO**.

Bits [31:18]

Reserved, RES0.

LASTFAILED, bit [17]

When FEAT_TME is implemented:

Indicates transaction failure or cancellation.

| LASTFAILED | Meaning |
|------------|---|
| 0b0 | Indicates that no transactions in a non-prohibited region have failed or been canceled between the previous Branch record and this Branch record. |
| 0b1 | Indicates that at least one transaction in a non-prohibited region has failed or been canceled between the previous Branch record and this Branch record. |

The value in this field is only valid when BRBINF<n>_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINF<n>_EL1.VALID == 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RES0.

T, bit [16]

When FEAT_TME is implemented:

Transactional state.

| T | Meaning |
|-----|--|
| 0b0 | The branch or exception was not executed in Transactional state. |
| 0b1 | The branch or exception was executed in Transactional state. |

The value in this field is only valid when BRBINF<n>_EL1.VALID == 0b10 or BRBINF<n>_EL1.VALID == 0b11.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- **RES0** if any of the following are true:
 - BRBINF<n>_EL1.VALID == 0b00
 - BRBINF<n>_EL1.VALID == 0b01
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [15:14]

Reserved, RES0.

TYPE, bits [13:8]

Branch type.

| TYPE | Meaning |
|----------|--|
| 0b000000 | Unconditional direct branch, excluding Branch with link. |
| 0b000001 | Indirect branch, excluding Branch with link, Return from subroutine, and Exception return. |
| 0b000010 | Direct Branch with link. |
| 0b000011 | Indirect Branch with link. |
| 0b000101 | Return from subroutine. |
| 0b000111 | Exception return. |
| 0b001000 | Conditional direct branch. |
| 0b100001 | Debug halt. |
| 0b100010 | Call. |
| 0b100011 | Trap. |
| 0b100100 | SError. |
| 0b100110 | Instruction debug. |
| 0b100111 | Data debug. |
| 0b101010 | Alignment. |
| 0b101011 | Inst Fault. |
| 0b101100 | Data Fault. |
| 0b101110 | IRQ. |
| 0b101111 | FIQ. |
| 0b111001 | Debug State Exit. |

All other values are reserved.

The value in this field is only valid when BRBINF<n>_EL1.VALID != 0b00.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When BRBINF<n>_EL1.VALID == 0b00, access to this field is **RES0**.
- Otherwise, access to this field is **RO**.

EL, bits [7:6]

The Exception Level at the target address.

| EL | Meaning | Applies when |
|------|---------|-----------------------------------|
| 0b00 | EL0. | |
| 0b01 | EL1. | |
| 0b10 | EL2. | |
| 0b11 | EL3. | When FEAT_BRBEv1p1 is implemented |

The value in this field is only valid when BRBINF<n>_EL1.VALID == 0b11 or BRBINF<n>_EL1.VALID == 0b01.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- RES0** if any of the following are true:
 - BRBINF<n>_EL1.VALID == 0b00
 - BRBINF<n>_EL1.VALID == 0b10
- Otherwise, access to this field is **RO**.

MPRED, bit [5]

Branch mispredict.

| MPRED | Meaning |
|-------|--|
| 0b0 | Branch was correctly predicted or the result of the prediction was not captured. |
| 0b1 | Branch was incorrectly predicted. |

The value in this field is only valid when BRBINF<n>_EL1.VALID == 0b11 or BRBINF<n>_EL1.VALID == 0b10.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- RES0** if any of the following are true:
 - BRBINF<n>_EL1.VALID == 0b00
 - BRBINF<n>_EL1.VALID == 0b01
 - BRBINF<n>_EL1.TYPE[5] == 1
- Otherwise, access to this field is **RO**.

Bits [4:2]

Reserved, RES0.

VALID, bits [1:0]

The Branch record is valid.

| VALID | Meaning |
|-------|--|
| 0b00 | This Branch record is not valid. The values of following fields are not valid: <ul style="list-style-type: none"> • BRBTGT<n>_EL1.ADDRESS. • BRBSRC<n>_EL1.ADDRESS. • BRBINF<n>_EL1.LASTFAILED. • BRBINF<n>_EL1.T. • BRBINF<n>_EL1.EL. • BRBINF<n>_EL1.TYPE. • BRBINF<n>_EL1.CC. • BRBINF<n>_EL1.CCU. |
| 0b01 | This Branch record is valid. The values of following fields are not valid: <ul style="list-style-type: none"> • BRBSRC<n>_EL1.ADDRESS. • BRBINF<n>_EL1.T. • BRBINF<n>_EL1.MPRED. |
| 0b10 | This Branch record is valid. The values of following fields are not valid: <ul style="list-style-type: none"> • BRBTGT<n>_EL1.ADDRESS. • BRBINF<n>_EL1.EL. |
| 0b11 | This Branch record is valid. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing BRBINF<n>_EL1

BRBINF<n>_EL1 reads-as-zero if $n + (\text{BRBFCR_EL1.BANK} \times 32) \geq \text{BRBIDR0_EL1.NUMREC}$.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, BRBINF<n>_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-----------|
| 0b10 | 0b001 | 0b1000 | n[3:0] | n[4]:0b00 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.nBRBDATA == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return BRBINF_EL1[UInt(op2<2>:CRm<3:0>)];
        elsif PSTATE.EL == EL2 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
                UNDEFINED;
            elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
                UNDEFINED;
            elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
                elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.SystemAccessTrap(EL3, 0x18);
                    else
                        return BRBINF_EL1[UInt(op2<2>:CRm<3:0>)];
        elsif PSTATE.EL == EL3 then
            return BRBINF_EL1[UInt(op2<2>:CRm<3:0>)];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBSRCINJ_EL1, Branch Record Buffer Source Address Injection Register

The BRBSRCINJ_EL1 characteristics are:

Purpose

The source address of a Branch record for injection.

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBSRCINJ_EL1 are UNDEFINED.

Attributes

BRBSRCINJ_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | ADDRESS | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ADDRESS | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ADDRESS, bits [63:0]

Source virtual address of the Branch record.

When a direct write occurs with a value with ADDRESS bits [63:P] being other than all zeroes or all ones, an UNKNOWN value which is not all zeroes or all ones is written to bits [63:P]. P is defined as the virtual address size supported by the PE, as returned by VAMax(). The value in bits [P-1:0] are the value written.

When a direct write occurs with a value with ADDRESS bits [63:P] being all zeroes or all ones, the written value is written to bits [63:0], and a read of the register returns the written value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- RES0** if any of the following are true:
 - BRBINFINJ_EL1.VALID == 0b00
 - BRBINFINJ_EL1.VALID == 0b01
- Otherwise, access to this field is **RW**.

Accessing BRBSRCINJ_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, BRBSRCINJ_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.nBRBDATA == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBSRCINJ_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBSRCINJ_EL1;
elsif PSTATE.EL == EL3 then
    return BRBSRCINJ_EL1;

```

MSR BRBSRCINJ_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.nBRBDATA == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBSRCINJ_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBSRCINJ_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    BRBSRCINJ_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBSRC<n>_EL1, Branch Record Buffer Source Address Register <n>, n = 0 - 31

The BRBSRC<n>_EL1 characteristics are:

Purpose

The source address of Branch record n + ([BRBFCCR_EL1](#).BANK × 32).

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBSRC<n>_EL1 are UNDEFINED.

Attributes

BRBSRC<n>_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | ADDRESS | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ADDRESS | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ADDRESS, bits [63:0]

Source virtual address of the Branch record.

When an indirect write occurs with a value with ADDRESS bits [63:P] being other than all zeroes or all ones, an UNKNOWN value which is not all zeroes or all ones is written to bits [63:P]. P is defined as the virtual address size supported by the PE, as returned by VAMax(). The value in bits [P-1:0] are the value written.

When an indirect write occurs with a value with ADDRESS bits [63:P] being all zeroes or all ones, the written value is written to bits [63:0], and a read of the register returns the written value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- RES0** if any of the following are true:
 - BRBINF<n>_EL1.VALID == 0b00
 - BRBINF<n>_EL1.VALID == 0b01
- Otherwise, access to this field is **RO**.

Accessing BRBSRC<n>_EL1

BRBSRC<n>_EL1 is RES0 if n + ([BRBFCCR_EL1](#).BANK × 32) >= [BRBIDR0_EL1](#).NUMREC.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, BRBSRC<n>_EL1

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-----------|
| 0b10 | 0b001 | 0b1000 | n[3:0] | n[4]:0b01 |
|------|-------|--------|--------|-----------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.nBRBDATA == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBSRC_EL1[UInt(op2<2>:CRm<3:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBSRC_EL1[UInt(op2<2>:CRm<3:0>)];
elsif PSTATE.EL == EL3 then
    return BRBSRC_EL1[UInt(op2<2>:CRm<3:0>)];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBTGTINJ_EL1, Branch Record Buffer Target Address Injection Register

The BRBTGTINJ_EL1 characteristics are:

Purpose

The target address of a Branch record for injection.

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBTGTINJ_EL1 are UNDEFINED.

Attributes

BRBTGTINJ_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | ADDRESS | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ADDRESS | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ADDRESS, bits [63:0]

Target virtual address of the Branch record.

When a direct write occurs with a value with ADDRESS bits [63:P] being other than all zeroes or all ones, an UNKNOWN value which is not all zeroes or all ones is written to bits [63:P]. P is defined as the virtual address size supported by the PE, as returned by VAMax(). The value in bits [P-1:0] are the value written.

When a direct write occurs with a value with ADDRESS bits [63:P] being all zeroes or all ones, the written value is written to bits [63:0], and a read of the register returns the written value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- RES0** if any of the following are true:
 - BRBINFINJ_EL1.VALID == 0b00
 - BRBINFINJ_EL1.VALID == 0b10
- Otherwise, access to this field is **RW**.

Accessing BRBTGTINJ_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, BRBTGTINJ_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.nBRBDATA == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBTGTINJ_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBTGTINJ_EL1;
elsif PSTATE.EL == EL3 then
    return BRBTGTINJ_EL1;

```

MSR BRBTGTINJ_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.nBRBDATA == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBTGTINJ_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBTGTINJ_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    BRBTGTINJ_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBTGT<n>_EL1, Branch Record Buffer Target Address Register <n>, n = 0 - 31

The BRBTGT<n>_EL1 characteristics are:

Purpose

The target address of Branch record n + ([BRBFCCR_EL1](#).BANK × 32).

Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBTGT<n>_EL1 are UNDEFINED.

Attributes

BRBTGT<n>_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | ADDRESS | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ADDRESS | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ADDRESS, bits [63:0]

Target virtual address of the Branch record.

When an indirect write occurs with a value with ADDRESS bits [63:P] being other than all zeroes or all ones, an UNKNOWN value which is not all zeroes or all ones is written to bits [63:P]. P is defined as the virtual address size supported by the PE, as returned by VAMax(). The value in bits [P-1:0] are the value written.

When an indirect write occurs with a value with ADDRESS bits [63:P] being all zeroes or all ones, the written value is written to bits [63:0], and a read of the register returns the written value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- RES0** if any of the following are true:
 - BRBINF<n>_EL1.VALID == 0b00
 - BRBINF<n>_EL1.VALID == 0b10
- Otherwise, access to this field is **RO**.

Accessing BRBTGT<n>_EL1

BRBTGT<n>_EL1 is RES0 if n + ([BRBFCCR_EL1](#).BANK × 32) >= [BRBIDRO_EL1](#).NUMREC.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, BRBTGT<n>_EL1

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-----------|
| 0b10 | 0b001 | 0b1000 | n[3:0] | n[4]:0b10 |
|------|-------|--------|--------|-----------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.nBRBDATA == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBTGT_EL1[UInt(op2<2>:CRm<3:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBTGT_EL1[UInt(op2<2>:CRm<3:0>)];
elsif PSTATE.EL == EL3 then
    return BRBTGT_EL1[UInt(op2<2>:CRm<3:0>)];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BRBTS_EL1, Branch Record Buffer Timestamp Register

The BRBTS_EL1 characteristics are:

Purpose

Captures the Timestamp value when branch recording becomes Paused.

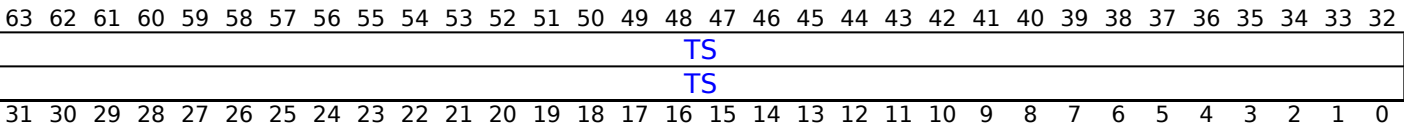
Configuration

This register is present only when FEAT_BRBE is implemented. Otherwise, direct accesses to BRBTS_EL1 are UNDEFINED.

Attributes

BRBTS_EL1 is a 64-bit register.

Field descriptions



TS, bits [63:0]

- Timestamp value at time of freezing.
- The reset behavior of this field is:
- On a Warm reset, this field resets to 0.

Accessing BRBTS_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, BRBTS_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.nBRBDATA == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elseif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBTS_EL1;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elseif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return BRBTS_EL1;
elseif PSTATE.EL == EL3 then
    return BRBTS_EL1;

```

MSR BRBTS_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b1001 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.nBRBDATA == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBTS_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE != '11' && SCR_EL3.NS == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.SBRBE == 'x0' && SCR_EL3.NS == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        BRBTS_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    BRBTS_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CCSIDR2_EL1, Current Cache Size ID Register 2

The CCSIDR2_EL1 characteristics are:

Purpose

Provides the information about the architecture of the currently selected cache from bits[63:32] of [CCSIDR_EL1](#).

Configuration

AArch64 System register CCSIDR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register [CCSIDR2\[31:0\]](#).

This register is present only when FEAT_CCIDX is implemented. Otherwise, direct accesses to CCSIDR2_EL1 are UNDEFINED.

In an implementation which does not support AArch32 at EL1, it is IMPLEMENTATION DEFINED whether reading this register gives an UNKNOWN value or is UNDEFINED.

The implementation includes one CCSIDR2_EL1 for each cache that it can access. [CSSELR_EL1](#) selects which Cache Size ID Register is accessible.

Attributes

CCSIDR2_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | NumSets | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

NumSets, bits [23:0]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Accessing CCSIDR2_EL1

If [CSSELR_EL1](#).Level is programmed to a cache level that is not implemented, then on a read of the CCSIDR2_EL1 the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR2_EL1 read is treated as NOP.
- The CCSIDR2_EL1 read is UNDEFINED.
- The CCSIDR2_EL1 read returns an UNKNOWN value.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CCSIDR2_EL1

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b001 | 0b0000 | 0b0000 | 0b010 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID2 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.TID4 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return CCSIDR2_EL1;
    elsif PSTATE.EL == EL2 then
        return CCSIDR2_EL1;
    elsif PSTATE.EL == EL3 then
        return CCSIDR2_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CCSIDR_EL1, Current Cache Size ID Register

The CCSIDR_EL1 characteristics are:

Purpose

Provides information about the architecture of the currently selected cache.

Configuration

AArch64 System register CCSIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [CCSIDR\[31:0\]](#).

AArch64 System register CCSIDR_EL1 bits [63:32] are architecturally mapped to AArch32 System register [CCSIDR2\[31:0\]](#).

The implementation includes one CCSIDR_EL1 for each cache that it can access. [CSSELR_EL1](#) selects which Cache Size ID Register is accessible.

Attributes

CCSIDR_EL1 is a 64-bit register.

Field descriptions

When FEAT_CCIDX is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|----|----|----|----|----|----|----|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------------------|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | NumSets | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | Associativity | | | | | | | | | | | | | | | | | | | | | | LineSize | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits [63:56]

Reserved, RES0.

NumSets, bits [55:32]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Bits [31:24]

Reserved, RES0.

Associativity, bits [23:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

($\log_2(\text{Number of bytes in cache line})$) - 4. For example:

- For a line length of 16 bytes: $\log_2(16) = 4$, LineSize entry = 0. This is the minimum line length.
- For a line length of 32 bytes: $\log_2(32) = 5$, LineSize entry = 1.

When FEAT_MTE2 is implemented and enabled, where a cache only holds Allocation tags, this field is RES0.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------------|----|----|----|----|----|----|----|----------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| UNKNOWN | | | | NumSets | | | | | | | | | | | | | | | | Associativity | | | | | | | | LineSize | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits [63:32]

Reserved, RES0.

Bits [31:28]

Reserved, UNKNOWN.

NumSets, bits [27:13]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Associativity, bits [12:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

($\log_2(\text{Number of bytes in cache line})$) - 4. For example:

- For a line length of 16 bytes: $\log_2(16) = 4$, LineSize entry = 0. This is the minimum line length.
- For a line length of 32 bytes: $\log_2(32) = 5$, LineSize entry = 1.

Accessing CCSIDR_EL1

If [CSSELR_EL1](#).Level is programmed to a cache level that is not implemented, then on a read of the CCSIDR_EL1 the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR_EL1 read is treated as NOP.

- The CCSIDR_EL1 read is UNDEFINED.
- The CCSIDR_EL1 read returns an UNKNOWN value.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CCSIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b001 | 0b0000 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID2 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.TID4 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.CCSIDR_EL1 == '1'
then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return CCSIDR_EL1;
    elsif PSTATE.EL == EL2 then
        return CCSIDR_EL1;
    elsif PSTATE.EL == EL3 then
        return CCSIDR_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CFP RCTX, Control Flow Prediction Restriction by Context

The CFP RCTX characteristics are:

Purpose

Control Flow Prediction Restriction by Context applies to all Control Flow Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

Control flow predictions determined by the actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control speculative execution occurring after the instruction is complete and synchronized.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

Configuration

This instruction is present only when FEAT_SPECRES is implemented. Otherwise, direct accesses to CFP RCTX are UNDEFINED.

Attributes

CFP RCTX is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|------|----|----|----|----|----|----|----|----|-------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | GVMID | VMID | | | | | | | | | | | | | | | | |
| RES0 | | | | NSENS | EL | RES0 | | | | | | | | | GASID | ASID | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:49]

Reserved, RES0.

GVMID, bit [48]

Execution of this instruction applies to all VMIDs or a specified VMID.

| GVMID | Meaning |
|-------|---|
| 0b0 | Applies to specified VMID for an EL0 or EL1 target execution context. |
| 0b1 | Applies to all VMIDs for an EL0 or EL1 target execution context. |

For target execution contexts other than EL0 or EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

VMID, bits [47:32]

Only applies when bit[48] is 0 and the target execution context is either:

- EL1.
- EL0 when ([HCR_EL2.E2H==0](#) or [HCR_EL2.TGE==0](#)).

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0 and ([HCR_EL2.E2H==0](#) or [HCR_EL2.TGE==0](#)), this field is treated as the current VMID.

When the instruction is executed at EL0 and ([HCR_EL2.E2H==1](#) and [HCR_EL2.TGE==1](#)), this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

If the implementation supports 16 bits of VMID, then the upper 8 bits of the VMID must be written to 0 by software when the context being affected only uses 8 bits.

Bits [31:28]

Reserved, RES0.

NSE, bit [27]

When FEAT_RME is implemented:

Together with the NS field, selects the Security state.

For a description of the values derived by evaluating NS and NSE together, see CFP_RCTX.NS.

Otherwise:

Reserved, RES0.

NS, bit [26]

When FEAT_RME is implemented:

Together with the NSE field, selects the Security state. Defined values are:

| NSE | NS | Meaning |
|-----|-----|-------------|
| 0b0 | 0b0 | Secure. |
| 0b0 | 0b1 | Non-secure. |
| 0b1 | 0b0 | Root. |
| 0b1 | 0b1 | Realm. |

Some Effective values are determined by the current Security state:

- When executed in Secure state, the Effective value of NSE is 0.
- When executed in Non-secure state, the Effective value of {NSE, NS} is {0, 1}.
- When executed in Realm state, the Effective value of {NSE, NS} is {1, 1}.

An instruction with an EL field that has a value other than 0b11 (EL3) is treated as a NOP when executed at EL3 with CFP_RCTX.{NSE, NS} == {1, 0}.

Otherwise:

Security State. Defined values are:

| NS | Meaning |
|-----|-------------------|
| 0b0 | Secure state. |
| 0b1 | Non-secure state. |

When executed in Non-secure state, the Effective value of NS is 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

| EL | Meaning |
|------|---------|
| 0b00 | EL0. |
| 0b01 | EL1. |
| 0b10 | EL2. |
| 0b11 | EL3. |

If the instruction is executed at an Exception level lower than the specified level, this instruction is treated as a NOP.

Bits [23:17]

Reserved, RES0.

GASID, bit [16]

Execution of this instruction applies to all ASIDs or a specified ASID.

| GASID | Meaning |
|-------|--|
| 0b0 | Applies to specified ASID for an EL0 target execution context. |
| 0b1 | Applies to all ASID for an EL0 target execution context. |

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field has an Effective value of 0.

ASID, bits [15:0]

Only applies for an EL0 target execution context and when bit[16] is 0.

Otherwise, this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being affected only uses 8 bits.

Executing the CFP RCTX instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

CFP RCTX, <Xt>

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b0011 | 0b100 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.CFPRCTX == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.EnRCTX == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.RestrictPrediction(X[t], RestrictType_ControlFlow);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.NV == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.CFPRCTX == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.RestrictPrediction(X[t], RestrictType_ControlFlow);
    elsif PSTATE.EL == EL2 then
        AArch64.RestrictPrediction(X[t], RestrictType_ControlFlow);
    elsif PSTATE.EL == EL3 then
        AArch64.RestrictPrediction(X[t], RestrictType_ControlFlow);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CLIDR_EL1, Cache Level ID Register

The CLIDR_EL1 characteristics are:

Purpose

Identifies the type of cache, or caches, that are implemented at each level and can be managed using the architected cache maintenance instructions that operate by set/way, up to a maximum of seven levels. Also identifies the Level of Coherence (LoC) and Level of Unification (LoU) for the cache hierarchy.

Configuration

AArch64 System register CLIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [CLIDR\[31:0\]](#).

Attributes

CLIDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|------|----|-----|----|-------|----|--------|----|--------|----|--------|----|--------|----|--------|--------|--------|--------|--------|--------|--------|--------|-----|----|----|----|----|----|----|----|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | |
| RES0 | | | | | | | | | | | | | | | | | Ttype7 | Ttype6 | Ttype5 | Ttype4 | Ttype3 | Ttype2 | Ttype1 | ICB | | | | | | | | | | |
| ICB | | LoUU | | LoC | | LoUIS | | Ctype7 | | Ctype6 | | Ctype5 | | Ctype4 | | Ctype3 | | Ctype2 | | Ctype1 | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | |

Bits [63:47]

Reserved, RES0.

Ttype<n>, bits [2(n-1)+34:2(n-1)+33], for n = 7 to 1

When FEAT_MTE2 is implemented:

Tag cache type. Indicate the type of cache that is implemented and can be managed using the architected cache maintenance instructions that operate by set/way at each level, from Level 1 up to a maximum of seven levels of cache hierarchy.

| Ttype<n> | Meaning |
|----------|--|
| 0b00 | No Tag Cache. |
| 0b01 | Separate Allocation Tag Cache. |
| 0b10 | Unified Allocation Tag and Data cache, Allocation Tags and Data in unified lines. |
| 0b11 | Unified Allocation Tag and Data cache, Allocation Tags and Data in separate lines. |

Otherwise:

Reserved, RES0.

ICB, bits [32:30]

Inner cache boundary. This field indicates the boundary for caching Inner Cacheable memory regions.

| ICB | Meaning |
|-------|--|
| 0b000 | Not disclosed by this mechanism. |
| 0b001 | L1 cache is the highest Inner Cacheable level. |
| 0b010 | L2 cache is the highest Inner Cacheable level. |
| 0b011 | L3 cache is the highest Inner Cacheable level. |
| 0b100 | L4 cache is the highest Inner Cacheable level. |
| 0b101 | L5 cache is the highest Inner Cacheable level. |
| 0b110 | L6 cache is the highest Inner Cacheable level. |
| 0b111 | L7 cache is the highest Inner Cacheable level. |

LoUU, bits [29:27]

Level of Unification Uniprocessor for the cache hierarchy.

Note

When FEAT_S2FWB is implemented, the architecture requires that this field is zero so that no levels of data cache need to be cleaned in order to manage coherency with instruction fetches.

LoC, bits [26:24]

Level of Coherence for the cache hierarchy.

LoUIS, bits [23:21]

Level of Unification Inner Shareable for the cache hierarchy.

Note

When FEAT_S2FWB is implemented, the architecture requires that this field is zero so that no levels of data cache need to be cleaned in order to manage coherency with instruction fetches.

Ctype<n>, bits [3(n-1)+2:3(n-1)], for n = 7 to 1

Cache Type fields. Indicate the type of cache that is implemented and can be managed using the architected cache maintenance instructions that operate by set/way at each level, from Level 1 up to a maximum of seven levels of cache hierarchy. Possible values of each field are:

| Ctype<n> | Meaning |
|----------|---------------------------------------|
| 0b000 | No cache. |
| 0b001 | Instruction cache only. |
| 0b010 | Data cache only. |
| 0b011 | Separate instruction and data caches. |
| 0b100 | Unified cache. |

All other values are reserved.

If software reads the Cache Type fields from Ctype1 upwards, once it has seen a value of 000, no caches that can be managed using the architected cache maintenance instructions that operate by set/way exist at further-out levels of the hierarchy. So, for example, if Ctype3 is the first Cache Type field with a value of 000, the values of Ctype4 to Ctype7 must be ignored.

Accessing CLIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CLIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b001 | 0b0000 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID2 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && HCR_EL2.TID4 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.CLIDR_EL1 == '1'
then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return CLIDR_EL1;
    elseif PSTATE.EL == EL2 then
        return CLIDR_EL1;
    elseif PSTATE.EL == EL3 then
        return CLIDR_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTFRQ_EL0, Counter-timer Frequency register

The CNTFRQ_EL0 characteristics are:

Purpose

This register is provided so that software can discover the frequency of the system counter. It must be programmed with this value as part of system initialization. The value of the register is not interpreted by hardware.

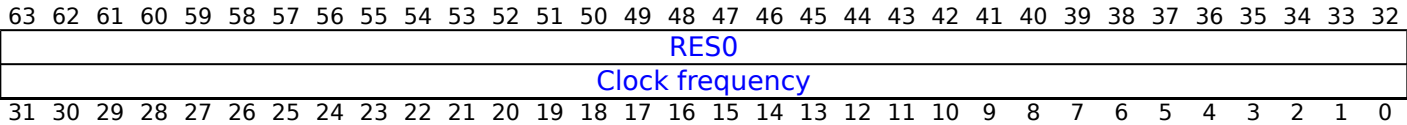
Configuration

AArch64 System register CNTFRQ_EL0 bits [31:0] are architecturally mapped to AArch32 System register [CNTFRQ\[31:0\]](#).

Attributes

CNTFRQ_EL0 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

Bits [31:0]

Clock frequency. Indicates the system counter clock frequency, in Hz.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTFRQ_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTFRQ_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.<EL0PCTEN,EL0VCTEN> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.<EL0PCTEN,EL0VCTEN> == '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CNTFRQ_EL0;
elseif PSTATE.EL == EL1 then
    return CNTFRQ_EL0;
elseif PSTATE.EL == EL2 then
    return CNTFRQ_EL0;
elseif PSTATE.EL == EL3 then
    return CNTFRQ_EL0;

```

MSR CNTFRQ_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0000 | 0b000 |

```

if IsHighestEL(PSTATE.EL) then
    CNTFRQ_EL0 = X[t];
else
    UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHCTL_EL2, Counter-timer Hypervisor Control register

The CNTHCTL_EL2 characteristics are:

Purpose

Controls the generation of an event stream from the physical counter, and access from EL1 to the physical counter and the EL1 physical timer.

Configuration

AArch64 System register CNTHCTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register [CNTHCTL\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

CNTHCTL_EL2 is a 64-bit register.

Field descriptions

When FEAT_VHE is implemented and HCR_EL2.E2H == 1:

| | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----------|----------|--------|----------|----------|---------|--------|-----|---------|---------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | |
| | | | | | | | | | | | | | | | | | | RES0 | | | | |
| RES0 | | | | | | | | | | | | CNTPMASK | CNTVMASK | EVNTIS | EL1NVVCT | EL1NVPCT | EL1TVCT | EL1TVT | ECV | EL1PTEN | EL1PCTE | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | |

Bits [63:20]

Reserved, RES0.

CNTPMASK, bit [19]

When FEAT_RME is implemented:

| CNTPMASK | Meaning |
|----------|---|
| 0b0 | This control has no affect on CNTP_CTL_EL0.IMASK . |
| 0b1 | CNTP_CTL_EL0.IMASK behaves as if set to 1 for all purposes other than a direct read of the field. |

This bit is RES0 in Non-secure and Secure state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CNTVMASK, bit [18]**When FEAT_RME is implemented:**

| CNTVMASK | Meaning |
|----------|--|
| 0b0 | This control has no affect on CNTV_CTL_EL0.IMASK . CNTV_CTL_EL0.IMASK behaves as if set to 1 for all purposes other than a direct read of the field. |
| 0b1 | |

This bit is RES0 in Non-secure and Secure state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVNTIS, bit [17]**When FEAT_ECV is implemented:**

Controls the scale of the generation of the event stream.

| EVNTIS | Meaning |
|--------|---|
| 0b0 | The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0 [15:0]. |
| 0b1 | The CNTHCTL_EL2.EVNTI field applies to CNTPCT_EL0 [23:8]. |

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVVCT, bit [16]**When FEAT_ECV is implemented:**

Traps EL1 accesses to the specified EL1 virtual timer registers using the EL02 descriptors to EL2, when EL2 is enabled for the current Security state.

| EL1NVVCT | Meaning |
|----------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | If ((HCR_EL2.E2H ==1 && HCR_EL2.TGE ==1) HCR_EL2.NV2 ==0 HCR_EL2.NV1 ==1 HCR_EL2.NV ==0), this control does not cause any instructions to be trapped. If ((HCR_EL2.E2H ==0 HCR_EL2.TGE ==0) && HCR_EL2.NV2 ==1 && HCR_EL2.NV1 ==0 && HCR_EL2.NV ==1), then EL1 accesses to CNTV_CTL_EL02 and CNTV_CVAL_EL02 are trapped to EL2. |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVPCT, bit [15]

When FEAT_ECV is implemented:

Traps EL1 accesses to the specified EL1 physical timer registers using the EL02 descriptors to EL2, when EL2 is enabled for the current Security state.

| EL1NVPCT | Meaning |
|----------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | If ((HCR_EL2.E2H ==1 && HCR_EL2.TGE ==1) HCR_EL2.NV2 ==0 HCR_EL2.NV1 ==1 HCR_EL2.NV ==0), this control does not cause any instructions to be trapped. If (HCR_EL2.E2H ==0 HCR_EL2.TGE ==0) && HCR_EL2.NV2 ==1 && HCR_EL2.NV1 ==0 && HCR_EL2.NV ==1, then EL1 accesses to CNTP_CTL_EL02 and CNTP_CVAL_EL02, are trapped to EL2. |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVCT, bit [14]

When FEAT_ECV is implemented:

Traps EL0 and EL1 accesses to the EL1 virtual counter registers to EL2, when EL2 is enabled for the current Security state.

| EL1TVCT | Meaning |
|---------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | If HCR_EL2.{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped. If HCR_EL2.E2H is 0 or HCR_EL2.TGE is 0, then: <ul style="list-style-type: none"> In AArch64 state, traps EL0 and EL1 accesses to CNTVCT_EL0 to EL2, unless they are trapped by CNTKCTL_EL1.EL0VCTEN. In AArch32 state, traps EL0 and EL1 accesses to CNTVCT to EL2, unless they are trapped by CNTKCTL_EL1.EL0VCTEN or CNTKCTL.PL0VCTEN. |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVT, bit [13]**When FEAT_ECV is implemented:**

Traps EL0 and EL1 accesses to the EL1 virtual timer registers to EL2, when EL2 is enabled for the current Security state.

| EL1TVT | Meaning |
|--------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | If HCR_EL2 .{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped. If HCR_EL2 .E2H is 0 or HCR_EL2 .TGE is 0, then: <ul style="list-style-type: none"> In AArch64 state, traps EL0 and EL1 accesses to CNTV_CTL_EL0, CNTV_CVAL_EL0, and CNTV_TVAL_EL0 to EL2, unless they are trapped by CNTKCTL_EL1.ELOVTEN. In AArch32 state, traps EL0 and EL1 accesses to CNTV_CTL, CNTV_CVAL, and CNTV_TVAL to EL2, unless they are trapped by CNTKCTL_EL1.ELOVTEN or CNTKCTL.PLOVTEN. |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ECV, bit [12]**When FEAT_ECV is implemented:**

Enables the Enhanced Counter Virtualization functionality registers.

| ECV | Meaning |
|-----|--|
| 0b0 | Enhanced Counter Virtualization functionality is disabled. |
| 0b1 | When HCR_EL2 .{E2H, TGE} == {1, 1} or SCR_EL3 .{NS, EEL2} == {0, 0}, then Enhanced Counter Virtualization functionality is disabled. When SCR_EL3 .NS or SCR_EL3 .EEL2 are 1, and HCR_EL2 .E2H or HCR_EL2 .TGE are 0, then Enhanced Counter Virtualization functionality is enabled when EL2 is enabled for the current Security state. This means that: <ul style="list-style-type: none"> An MRS to CNTPCT_EL0 from either EL0 or EL1 that is not trapped will return the value (PCount<63:0> - CNTPOFF_EL2<63:0>). The EL1 physical timer interrupt is triggered when ((PCount<63:0> - CNTPOFF_EL2<63:0>) - PCVal<63:0>) is greater than or equal to 0. PCount<63:0> is the physical count returned when CNTPCT_EL0 is read from EL2 or EL3. PCVal<63:0> is the EL1 physical timer compare value for this timer. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1PTEN, bit [11]

When [HCR_EL2.TGE](#) is 0, traps EL0 and EL1 accesses to the E1 physical timer registers to EL2 when EL2 is enabled in the current Security state.

| EL1PTEN | Meaning |
|---------|---|
| 0b0 | From AArch64 state: EL0 and EL1 accesses to the CNTP_CTL_EL0 , CNTP_CVAL_EL0 , and CNTP_TVAL_EL0 are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1.ELOPTEN . From AArch32 state: EL0 and EL1 accesses to the CNTP_CTL , CNTP_CVAL , and CNTP_TVAL are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1.ELOPTEN or CNTKCTL.PL0PTEN . |
| 0b1 | This control does not cause any instructions to be trapped. |

When [HCR_EL2.TGE](#) is 1, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL1PCTEN, bit [10]

When [HCR_EL2.TGE](#) is 0, traps EL0 and EL1 accesses to the EL1 physical counter register to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to [CNTPCT_EL0](#) are trapped to EL2, reported using EC syndrome value 0x18.
- In AArch32 state, MRRC or MCRR accesses to [CNTPCT](#) are trapped to EL2, reported using EC syndrome value 0x04.

| EL1PCTEN | Meaning |
|----------|--|
| 0b0 | From AArch64 state: EL0 and EL1 accesses to the CNTPCT_EL0 are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1.EL0PCTEN . From AArch32 state: EL0 and EL1 accesses to the CNTPCT are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1.EL0PCTEN or CNTKCTL.PL0PCTEN . |
| 0b1 | This control does not cause any instructions to be trapped. |

When [HCR_EL2.TGE](#) is 1, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ELOPTEN, bit [9]

When [HCR_EL2.TGE](#) is 0, this control does not cause any instructions to be trapped.

When [HCR_EL2.TGE](#) is 1, traps EL0 accesses to the physical timer registers to EL2.

| ELOPTEN | Meaning |
|---------|--|
| 0b0 | EL0 using AArch64: EL0 accesses to the CNTP_CTL_EL0 , CNTP_CVAL_EL0 , and CNTP_TVAL_EL0 registers are trapped to EL2. EL0 using AArch32: EL0 accesses to the CNTP_CTL , CNTP_CVAL and CNTP_TVAL registers are trapped to EL2. |
| 0b1 | This control does not cause any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ELOVTEN, bit [8]

When [HCR_EL2.TGE](#) is 0, this control does not cause any instructions to be trapped.

When [HCR_EL2.TGE](#) is 1, traps EL0 accesses to the virtual timer registers to EL2.

| ELOVTEN | Meaning |
|---------|--|
| 0b0 | EL0 using AArch64: EL0 accesses to the CNTV_CTL_EL0 , CNTV_CVAL_EL0 , and CNTV_TVAL_EL0 registers are trapped to EL2. EL0 using AArch32: EL0 accesses to the CNTV_CTL , CNTV_CVAL , and CNTV_TVAL registers are trapped to EL2. |
| 0b1 | This control does not cause any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTI, bits [7:4]

Selects which bit of [CNTPCT_EL0](#), as seen from EL2, is the trigger for the event stream generated from that counter when that stream is enabled.

If FEAT_ECV is implemented, and CNTHCTL_EL2.EVNTIS is 1, this field selects a trigger bit in the range 8 to 23 of [CNTPCT_EL0](#).

Otherwise, this field selects a trigger bit in the range 0 to 15 of [CNTPCT_EL0](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the [CNTPCT_EL0](#) trigger bit, as seen from EL2 and defined by EVNTI, generates an event when the event stream is enabled.

| EVNTDIR | Meaning |
|---------|---|
| 0b0 | A 0 to 1 transition of the trigger bit triggers an event. |
| 0b1 | A 1 to 0 transition of the trigger bit triggers an event. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

Enables the generation of an event stream from [CNTPCT_EL0](#) as seen from EL2.

| EVNTEN | Meaning |
|--------|----------------------------|
| 0b0 | Disables the event stream. |
| 0b1 | Enables the event stream. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ELOVCTEN, bit [1]

When [HCR_EL2.TGE](#) is 0, this control does not cause any instructions to be trapped.

When [HCR_EL2.TGE](#) is 1, traps EL0 accesses to the frequency register and virtual counter register to EL2.

| EL0VCTEN | Meaning |
|----------|---|
| 0b0 | <p>EL0 using AArch64: EL0 accesses to the CNTVCT_EL0 are trapped to EL2.</p> <p>EL0 using AArch64: EL0 accesses to the CNTFRQ_EL0 register are trapped to EL2, if CINTHCTL_EL2.EL0PCTEN is also 0.</p> <p>EL0 using AArch32: EL0 accesses to the CNTVCT are trapped to EL2.</p> <p>EL0 using AArch32: EL0 accesses to the CNTFRQ register are trapped to EL2, if CINTHCTL.EL0PCTEN is also 0.</p> |
| 0b1 | This control does not cause any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ELOPCTEN, bit [0]

When [HCR_EL2.TGE](#) is 0, this control does not cause any instructions to be trapped.

When [HCR_EL2.TGE](#) is 1, traps EL0 accesses to the frequency register and physical counter register to EL2.

| EL0PCTEN | Meaning |
|----------|---|
| 0b0 | <p>EL0 using AArch64: EL0 accesses to the CNTPCT_EL0 are trapped to EL2.</p> <p>EL0 using AArch64: EL0 accesses to the CNTFRQ_EL0 register are trapped to EL2, if CNTHCTL_EL2.EL0VCTEN is also 0.</p> <p>EL0 using AArch32: EL0 accesses to the CNTPCT are trapped to EL2.</p> <p>EL0 using AArch32: EL0 accesses to the CNTFRQ and register are trapped to EL2, if CNTHCTL_EL2.EL0VCTEN is also 0.</p> |
| 0b1 | This control does not cause any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

| | | | | | | | | | | | | | | | | |
|--------------------------|----------|----------|--------|---------|----------|---------|--------|-----|------|-------|----|----|----|----|----|----|
| 636261605958575655545352 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 |
| RES0 | | | | | | | | | | | | | | | | |
| RES0 | CNTPMASK | CNTVMASK | EVNTIS | ELINVCT | ELINVPCT | EL1TVCT | EL1TVT | ECV | RES0 | EVNTI | | | | | | |
| 313029282726252423222120 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 |

This format applies in all Armv8.0 implementations, and it also contains a description of the behavior when EL3 is implemented and EL2 is not implemented.

Bits [63:20]

Reserved, RES0.

CNTPMASK, bit [19]

When FEAT_RME is implemented:

| CNTPMASK | Meaning |
|----------|--|
| 0b0 | This control has no affect on CNTP_CTL_ELO .IMASK. |
| 0b1 | CNTP_CTL_ELO .IMASK behaves as if set to 1 for all purposes other than a direct read of the field. |

This bit is RES0 in Non-secure and Secure state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CNTVMASK, bit [18]

When FEAT_RME is implemented:

| CNTVMASK | Meaning |
|----------|--|
| 0b0 | This control has no affect on CNTV_CTL_ELO .IMASK. |
| 0b1 | CNTV_CTL_ELO .IMASK behaves as if set to 1 for all purposes other than a direct read of the field. |

This bit is RES0 in Non-secure and Secure state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVNTIS, bit [17]

When FEAT_ECV is implemented:

Controls the scale of the generation of the event stream.

| EVNTIS | Meaning |
|--------|---|
| 0b0 | The CNTHCTL_EL2.EVNTI field applies to CNTPCT_ELO [15:0]. |
| 0b1 | The CNTHCTL_EL2.EVNTI field applies to CNTPCT_ELO [23:8]. |

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVVCT, bit [16]**When FEAT_ECV is implemented:**

Traps EL1 accesses to the specified EL1 virtual timer registers using the EL02 descriptors to EL2, when EL2 is enabled for the current Security state.

| EL1NVVCT | Meaning |
|----------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | If ((HCR_EL2.E2H==1 && HCR_EL2.TGE==1) HCR_EL2.NV2==0 HCR_EL2.NV1==1 HCR_EL2.NV==0), this control does not cause any instructions to be trapped. If ((HCR_EL2.E2H==0 HCR_EL2.TGE==0) && HCR_EL2.NV2==1 && HCR_EL2.NV1==0 && HCR_EL2.NV==1), then EL1 accesses to CNTV_CTL_EL02 and CNTV_CVAL_EL02 are trapped to EL2. |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1NVPCT, bit [15]**When FEAT_ECV is implemented:**

Traps EL1 accesses to the specified EL1 physical timer registers using the EL02 descriptors to EL2, when EL2 is enabled for the current Security state.

| EL1NVPCT | Meaning |
|----------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | If ((HCR_EL2.E2H==1 && HCR_EL2.TGE==1) HCR_EL2.NV2==0 HCR_EL2.NV1==1 HCR_EL2.NV==0), this control does not cause any instructions to be trapped. If (HCR_EL2.E2H==0 HCR_EL2.TGE==0) && HCR_EL2.NV2==1 && HCR_EL2.NV1==0 && HCR_EL2.NV==1 , then EL1 accesses to CNTP_CTL_EL02 and CNTP_CVAL_EL02, are trapped to EL2. |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVCT, bit [14]**When FEAT_ECV is implemented:**

Traps EL0 and EL1 accesses to the EL1 virtual counter registers to EL2, when EL2 is enabled for the current Security state.

| EL1TVCT | Meaning |
|---------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | If HCR_EL2 .{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped. If HCR_EL2 .E2H is 0 or HCR_EL2 .TGE is 0, then: In AArch64 state, traps EL0 and EL1 accesses to CNTVCT_EL0 to EL2, unless they are trapped by CNTKCTL_EL1 .EL0VCTEN. In AArch32 state, traps EL0 and EL1 accesses to CNTVCT to EL2, unless they are trapped by CNTKCTL_EL1 .EL0VCTEN or CNTKCTL .PL0VCTEN. |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EL1TVT, bit [13]**When FEAT_ECV is implemented:**

Traps EL0 and EL1 accesses to the EL1 virtual timer registers to EL2, when EL2 is enabled for the current Security state.

| EL1TVT | Meaning |
|--------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | If HCR_EL2 .{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped. If HCR_EL2 .E2H is 0 or HCR_EL2 .TGE is 0, then: <ul style="list-style-type: none"> In AArch64 state, traps EL0 and EL1 accesses to CNTV_CTL_EL0, CNTV_CVAL_EL0, and CNTV_TVAL_EL0 to EL2, unless they are trapped by CNTKCTL_EL1.EL0VTEN. In AArch32 state, traps EL0 and EL1 accesses to CNTV_CTL, CNTV_CVAL, and CNTV_TVAL to EL2, unless they are trapped by CNTKCTL_EL1.EL0VTEN or CNTKCTL.PL0VTEN. |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 0 other than for the purpose of a direct read.

This control applies regardless of the value of the CNTHCTL_EL2.ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ECV, bit [12]**When FEAT_ECV is implemented:**

Enables the Enhanced Counter Virtualization functionality registers.

| ECV | Meaning |
|-----|--|
| 0b0 | Enhanced Counter Virtualization functionality is disabled. |
| 0b1 | When HCR_EL2 .{E2H, TGE} == {1, 1} or SCR_EL3 .{NS, EEL2} == {0, 0}, then Enhanced Counter Virtualization functionality is disabled. When SCR_EL3 .NS or SCR_EL3 .EEL2 are 1, and HCR_EL2 .E2H or HCR_EL2 .TGE are 0, then Enhanced Counter Virtualization functionality is enabled when EL2 is enabled for the current Security state. This means that: <ul style="list-style-type: none"> An MRS to CNTPCT_EL0 from either EL0 or EL1 that is not trapped will return the value (PCount<63:0> - CNTPOFF_EL2<63:0>). The EL1 physical timer interrupt is triggered when ((PCount<63:0> - CNTPOFF_EL2<63:0>) - PCVal<63:0>) is greater than or equal to 0. PCount is the physical count returned when CNTPCT_EL0 is read from EL2 or EL3. PCVal<63:0> is the EL1 physical timer compare value for this timer. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [11:8]

Reserved, RES0.

EVNTI, bits [7:4]

Selects which bit of [CNTPCT_EL0](#), as seen from EL2, is the trigger for the event stream generated from that counter when that stream is enabled.

If FEAT_ECV is implemented, and CNTHCTL_EL2.EVNTIS is 1, this field selects a trigger bit in the range 8 to 23 of [CNTPCT_EL0](#).

Otherwise, this field selects a trigger bit in the range 0 to 15 of [CNTPCT_EL0](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the [CNTPCT_EL0](#) trigger bit, as seen from EL2 and defined by EVNTI, generates an event when the event stream is enabled.

| EVNTDIR | Meaning |
|---------|---|
| 0b0 | A 0 to 1 transition of the trigger bit triggers an event. |
| 0b1 | A 1 to 0 transition of the trigger bit triggers an event. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

Enables the generation of an event stream from [CNTPCT_EL0](#) as seen from EL2.

| EVNTEN | Meaning |
|--------|----------------------------|
| 0b0 | Disables the event stream. |
| 0b1 | Enables the event stream. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL1PCEN, bit [1]

Traps EL0 and EL1 accesses to the EL1 physical timer registers to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to [CNTP_CTL_EL0](#), [CNTP_CVAL_EL0](#), [CNTP_TVAL_EL0](#) are trapped to EL2, reported using EC syndrome value 0x18.
- In AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2 reported using EC syndrome value 0x3 and MRRC and MCRR accesses are trapped to EL2, reported using EC syndrome value 0x04:
 - [CNTP_CTL](#), [CNTP_CVAL](#), [CNTP_TVAL](#).

| EL1PCEN | Meaning |
|---------|---|
| 0b0 | From AArch64 state: EL0 and EL1 accesses to the CNTP_CTL_EL0 , CNTP_CVAL_EL0 , and CNTP_TVAL_EL0 are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1.EL0PTEN . From AArch32 state: EL0 and EL1 accesses to the CNTP_CTL , CNTP_CVAL , and CNTP_TVAL are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1.EL0PTEN or CNTKCTL.PL0PTEN . |
| 0b1 | This control does not cause any instructions to be trapped. |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL1PCTEN, bit [0]

Traps EL0 and EL1 accesses to the EL1 physical counter register to EL2 when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to [CNTPCT_EL0](#) are trapped to EL2, reported using EC syndrome value 0x18.
- In AArch32 state, MRRC or MCRR accesses to [CNTPCT](#) are trapped to EL2, reported using EC syndrome value 0x04.

| EL1PCTEN | Meaning |
|----------|--|
| 0b0 | From AArch64 state: EL0 and EL1 accesses to the CNTPCT_EL0 are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1.EL0PCTEN . From AArch32 state: EL0 and EL1 accesses to the CNTPCT are trapped to EL2 when EL2 is enabled in the current Security state, unless they are trapped by CNTKCTL_EL1.EL0PCTEN or CNTKCTL.PL0PCTEN . |
| 0b1 | This control does not cause any instructions to be trapped. |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHCTL_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic CNTHCTL_EL2 or CNTKCTL_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHCTL_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHCTL_EL2;
elsif PSTATE.EL == EL3 then
    return CNTHCTL_EL2;

```

MSR CNTHCTL_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHCTL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CNTHCTL_EL2 = X[t];

```

MRS <Xt>, CNTKCTL_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1110 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    return CNTKCTL_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CNTHCTL_EL2;
    else
        return CNTKCTL_EL1;
elsif PSTATE.EL == EL3 then
    return CNTKCTL_EL1;

```

MSR CNTKCTL_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1110 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    CNTKCTL_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CNTHCTL_EL2 = X[t];
    else
        CNTKCTL_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    CNTKCTL_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHP_CTL_EL2, Counter-timer Hypervisor Physical Timer Control register

The CNTHP_CTL_EL2 characteristics are:

Purpose

Control register for the EL2 physical timer.

Configuration

AArch64 System register CNTHP_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register [CNTHP_CTL\[31:0\]](#).

This register is present only when EL3 is implemented or (EL3 is not implemented, EL2 is implemented and FEAT_SEL2 is not implemented). Otherwise, direct accesses to CNTHP_CTL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHP_CTL_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|--|--|--|-------|--|--------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ISTATUS | | | | IMASK | | ENABLE | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | |

Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTHP_TVAL_EL2](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHP_CTL_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic [CNTHP_CTL_EL2](#) or [CNTPT_CTL_EL0](#) are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHP_CTL_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHP_CTL_EL2;
elsif PSTATE.EL == EL3 then
    return CNTHP_CTL_EL2;

```

MSR CNTHP_CTL_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CTL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CNTHP_CTL_EL2 = X[t];

```

MRS <Xt>, CNTP_CTL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
        IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CTL_EL2;
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x180];
    else
        return CNTP_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CTL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL_EL0;
elsif PSTATE.EL == EL3 then
    return CNTP_CTL_EL0;

```

MSR CNTP_CTL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = X[t];
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHP_CTL_EL2 = X[t];
    else
        CNTP_CTL_EL0 = X[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x180] = X[t];
    else
        CNTP_CTL_EL0 = X[t];
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_CTL_EL2 = X[t];
    else
        CNTP_CTL_EL0 = X[t];
elseif PSTATE.EL == EL3 then
    CNTP_CTL_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHP_CVAL_EL2, Counter-timer Physical Timer CompareValue register (EL2)

The CNTHP_CVAL_EL2 characteristics are:

Purpose

Holds the compare value for the EL2 physical timer.

Configuration

AArch64 System register CNTHP_CVAL_EL2 bits [63:0] are architecturally mapped to AArch32 System register [CNTHP_CVAL\[63:0\]](#).

This register is present only when EL3 is implemented or (EL3 is not implemented, EL2 is implemented and FEAT_SEL2 is not implemented). Otherwise, direct accesses to CNTHP_CVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHP_CVAL_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

CompareValue, bits [63:0]

Holds the EL2 physical timer CompareValue.

When [CNTHP_CTL_EL2.ENABLE](#) is 1, the timer condition is met when ([CNTPTCT_EL0](#) - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTHP_CTL_EL2.ISTATUS](#) is set to 1.
- If [CNTHP_CTL_EL2.IMASK](#) is 0, an interrupt is generated.

When [CNTHP_CTL_EL2.ENABLE](#) is 0, the timer condition is not met, but [CNTPTCT_EL0](#) continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHP_CVAL_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic CNTHP_CVAL_EL2 or CNTPT_CVAL_EL0 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHP_CVAL_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHP_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    return CNTHP_CVAL_EL2;

```

MSR CNTHP_CVAL_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CVAL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CNTHP_CVAL_EL2 = X[t];

```

MRS <Xt>, CNTP_CVAL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CVAL_EL2;
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHP_CVAL_EL2;
    else
        return CNTP_CVAL_EL0;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x178];
    else
        return CNTP_CVAL_EL0;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CVAL_EL2;
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHP_CVAL_EL2;
    else
        return CNTP_CVAL_EL0;
elseif PSTATE.EL == EL3 then
    return CNTP_CVAL_EL0;

```

MSR CNTP_CVAL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = X[t];
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x178] = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHP_TVAL_EL2, Counter-timer Physical Timer TimerValue register (EL2)

The CNTHP_TVAL_EL2 characteristics are:

Purpose

Holds the timer value for the EL2 physical timer.

Configuration

AArch64 System register CNTHP_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register [CNTHP_TVAL\[31:0\]](#).

This register is present only when EL3 is implemented or (EL3 is not implemented, EL2 is implemented and FEAT_SEL2 is not implemented). Otherwise, direct accesses to CNTHP_TVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHP_TVAL_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TimerValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If [CNTHP_CTL_EL2.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTHP_CTL_EL2.ENABLE](#) is 1, the value returned is ([CNTHP_CVAL_EL2](#) - [CNTPCT_EL0](#)).

On a write of this register, [CNTHP_CVAL_EL2](#) is set to ([CNTPCT_EL0](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTHP_CTL_EL2.ENABLE](#) is 1, the timer condition is met when ([CNTPCT_EL0](#) - [CNTHP_CVAL_EL2](#)) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTHP_CTL_EL2.ISTATUS](#) is set to 1.
- If [CNTHP_CTL_EL2.IMASK](#) is 0, an interrupt is generated.

When [CNTHP_CTL_EL2.ENABLE](#) is 0, the timer condition is not met, but [CNTPCT_EL0](#) continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHP_TVAL_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic [CNTHP_TVAL_EL2](#) or [CNTPTVAL_EL0](#) are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHP_TVAL_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if CNTHP_CTL_EL2.ENABLE == '0' then
        return bits(64) UNKNOWN;
    else
        return CNTHP_CVAL_EL2 - PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if CNTHP_CTL_EL2.ENABLE == '0' then
        return bits(64) UNKNOWN;
    else
        return CNTHP_CVAL_EL2 - PhysicalCountInt();

```

MSR CNTHP_TVAL_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    CNTHP_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();

```

MRS <Xt>, CNTPTVAL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHPS_CVAL_EL2 - PhysicalCountInt();
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHP_CVAL_EL2 - PhysicalCountInt();
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
        if CNTP_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTP_CVAL_EL0 - PhysicalCountInt();
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
CNTHCTL_EL2.ECV == '1' then
        if CNTP_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTP_CVAL_EL0 - PhysicalCountInt();
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHPS_CVAL_EL2 - PhysicalCountInt();
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHP_CVAL_EL2 - PhysicalCountInt();
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTP_CVAL_EL0 - PhysicalCountInt();
elseif PSTATE.EL == EL3 then
    if CNTP_CTL_EL0.ENABLE == '0' then
        return bits(64) UNKNOWN;
    else

```

```
return CNTP_CVAL_EL0 - PhysicalCountInt();
```

MSR CNTP_TVAL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b000 |

```
if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
        IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
        CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
        CNTHCTL_EL2.ECV == '1' then
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    else
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHPS_CTL_EL2, Counter-timer Secure Physical Timer Control register (EL2)

The CNTHPS_CTL_EL2 characteristics are:

Purpose

Control register for the Secure EL2 physical timer.

Configuration

AArch64 System register CNTHPS_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register [CNTHPS_CTL\[31:0\]](#).

This register is present only when FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHPS_CTL_EL2 are UNDEFINED.

Attributes

CNTHPS_CTL_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|--|--|---------|--|-------|--|--------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ISTATUS | | IMASK | | ENABLE | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | |

Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the CNTHPS_CTL_EL2.ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the CNTHPS CTL EL2.ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTHPS_TVAL_EL2](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_CTL_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHPS_CTL_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        return CNTHPS_CTL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        return CNTHPS_CTL_EL2;

```

MSR CNTHPS_CTL_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0101 | 0b001 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        CNTHPS_CTL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHPS_CTL_EL2 = X[t];

```

MRS <Xt>, CNTP_CTL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CTL_EL2;
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x180];
    else
        return CNTP_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CTL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL_EL0;
elsif PSTATE.EL == EL3 then
    return CNTP_CTL_EL0;

```

MSR CNTP_CTL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = X[t];
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHPS_CTL_EL2 = X[t];
    else
        CNTP_CTL_EL0 = X[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x180] = X[t];
    else
        CNTP_CTL_EL0 = X[t];
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHPS_CTL_EL2 = X[t];
    else
        CNTP_CTL_EL0 = X[t];
elseif PSTATE.EL == EL3 then
    CNTP_CTL_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHPS_CVAL_EL2, Counter-timer Secure Physical Timer CompareValue register (EL2)

The CNTHPS_CVAL_EL2 characteristics are:

Purpose

Holds the compare value for the Secure EL2 physical timer.

Configuration

AArch64 System register CNTHPS_CVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register [CNTHPS_CVAL\[31:0\]](#).

This register is present only when EL2 is implemented and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHPS_CVAL_EL2 are UNDEFINED.

Attributes

CNTHPS_CVAL_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| | | | | | | | | | | | | | | | | | CompareValue | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | CompareValue | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

CompareValue, bits [63:0]

Holds the EL2 physical timer CompareValue.

When `CNTHPS_CTL_EL2.ENABLE` is 1, the timer condition is met when `(CNTPCT_EL0 - CompareValue)` is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTHPS_CTL_EL2](#).ISTATUS is set to 1.
- If [CNTHPS_CTL_EL2](#).IMASK is 0, an interrupt is generated.

When [CNTHPS_CTL_EL2.ENABLE](#) is 0, the timer condition is not met, but [CNTPCT_EL0](#) continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_CVAL_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHPS CVAL EL2

| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0101 | 0b010 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        return CNTHPS_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        return CNTHPS_CVAL_EL2;

```

MSR CNTHPS_CVAL_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0101 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = X[t];

```

MRS <Xt>, CNTP_CVAL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CVAL_EL2;
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHP_CVAL_EL2;
    else
        return CNTP_CVAL_EL0;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x178];
    else
        return CNTP_CVAL_EL0;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CVAL_EL2;
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHP_CVAL_EL2;
    else
        return CNTP_CVAL_EL0;
elseif PSTATE.EL == EL3 then
    return CNTP_CVAL_EL0;

```

MSR CNTP_CVAL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = X[t];
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x178] = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHPS_TVAL_EL2, Counter-timer Secure Physical Timer TimerValue register (EL2)

The CNTHPS_TVAL_EL2 characteristics are:

Purpose

Holds the timer value for the Secure EL2 physical timer.

Configuration

AArch64 System register CNTHPS_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register [CNTHPS_TVAL\[31:0\]](#).

This register is present only when EL2 is implemented and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHPS_TVAL_EL2 are UNDEFINED.

Attributes

CNTHPS_TVAL_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TimerValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If [CNTHPS_CTL_EL2.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTHPS_CTL_EL2.ENABLE](#) is 1, the value returned is ([CNTHPS_CVAL_EL2](#) - [CNTPCT_EL0](#)).

On a write of this register, [CNTHPS_CVAL_EL2](#) is set to ([CNTPCT_EL0](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTHPS_CTL_EL2.ENABLE](#) is 1, the timer condition is met when ([CNTPCT_EL0](#) - [CNTHPS_CVAL_EL2](#)) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTHPS_CTL_EL2.ISTATUS](#) is set to 1.
- If [CNTHPS_CTL_EL2.IMASK](#) is 0, an interrupt is generated.

When [CNTHPS_CTL_EL2.ENABLE](#) is 0, the timer condition is not met, but [CNTPCT_EL0](#) continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_TVAL_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHPS_TVAL_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHPS_CVAL_EL2 - PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHPS_CVAL_EL2 - PhysicalCountInt();

```

MSR CNTHPS_TVAL_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHPS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();

```

MRS <Xt>, CNTP_TVAL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHPS_CVAL_EL2 - PhysicalCountInt();
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHPS_CVAL_EL2 - PhysicalCountInt();
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
        if CNTP_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTP_CVAL_EL0 - PhysicalCountInt();
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
CNTHCTL_EL2.ECV == '1' then
            if CNTP_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2);
        else
            if CNTP_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTP_CVAL_EL0 - PhysicalCountInt();
    elseif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
            if CNTHPS_CTL_EL2.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTHPS_CVAL_EL2 - PhysicalCountInt();
        elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
            if CNTHPS_CTL_EL2.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTHPS_CVAL_EL2 - PhysicalCountInt();
        else
            if CNTP_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTP_CVAL_EL0 - PhysicalCountInt();
    elseif PSTATE.EL == EL3 then
        if CNTP_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTP_CVAL_EL0 - PhysicalCountInt();

```

```
return CNTP_CVAL_EL0 - PhysicalCountInt();
```

MSR CNTP_TVAL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
CNTHCTL_EL2.ECV == '1' then
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    else
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHV_CTL_EL2, Counter-timer Virtual Timer Control register (EL2)

The CNTHV_CTL_EL2 characteristics are:

Purpose

Control register for the EL2 virtual timer.

Configuration

AArch64 System register CNTHV_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register [CNTHV_CTL\[31:0\]](#).

This register is present only when FEAT_VHE is implemented and (EL3 is implemented or (EL3 is not implemented and FEAT_SEL2 is not implemented)). Otherwise, direct accesses to CNTHV_CTL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHV_CTL_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ISTATUSIMASKENABLE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTHV_TVAL_EL2](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHV_CTL_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic CNTHV_CTL_EL2 or CNTV_CTL_EL0 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHV_CTL_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHV_CTL_EL2;
elsif PSTATE.EL == EL3 then
    return CNTHV_CTL_EL2;

```

MSR CNTHV_CTL_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHV_CTL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CNTHV_CTL_EL2 = X[t];

```

MRS <Xt>, CNTV_CTL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
        IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CTL_EL2;
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x170];
    else
        return CNTV_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CTL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL_EL0;
elsif PSTATE.EL == EL3 then
    return CNTV_CTL_EL0;

```

MSR CNTV_CTL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t];
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_CTL_EL2 = X[t];
    else
        CNTV_CTL_EL0 = X[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x170] = X[t];
    else
        CNTV_CTL_EL0 = X[t];
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHV_CTL_EL2 = X[t];
    else
        CNTV_CTL_EL0 = X[t];
elseif PSTATE.EL == EL3 then
    CNTV_CTL_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHV_CVAL_EL2, Counter-timer Virtual Timer CompareValue register (EL2)

The CNTHV_CVAL_EL2 characteristics are:

Purpose

Holds the compare value for the EL2 virtual timer.

Configuration

AArch64 System register CNTHV_CVAL_EL2 bits [63:0] are architecturally mapped to AArch32 System register [CNTHV_CVAL\[63:0\]](#).

This register is present only when FEAT_VHE is implemented and (EL3 is implemented or (EL3 is not implemented and FEAT_SEL2 is not implemented)). Otherwise, direct accesses to CNTHV_CVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHV_CVAL_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

CompareValue, bits [63:0]

Holds the EL2 virtual timer CompareValue.

When [CNTHV_CTL_EL2](#).ENABLE is 1, the timer condition is met when ([CNTVCT_EL0](#) - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTHV_CTL_EL2](#).ISTATUS is set to 1.
- If [CNTHV_CTL_EL2](#).IMASK is 0, an interrupt is generated.

When [CNTHV_CTL_EL2](#).ENABLE is 0, the timer condition is not met, but [CNTVCT_EL0](#) continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHV_CVAL_EL2

When [HCR_EL2](#).E2H is 1, without explicit synchronization, access from EL2 using the mnemonic CNTHV_CVAL_EL2 or CNTV_CVAL_EL0 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHV_CVAL_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHV_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    return CNTHV_CVAL_EL2;

```

MSR CNTHV_CVAL_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHV_CVAL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CNTHV_CVAL_EL2 = X[t];

```

MRS <Xt>, CNTV_CVAL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CVAL_EL2;
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL_EL0;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x168];
    else
        return CNTV_CVAL_EL0;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CVAL_EL2;
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL_EL0;
elseif PSTATE.EL == EL3 then
    return CNTV_CVAL_EL0;

```

MSR CNTV_CVAL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t];
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x168] = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL3 then
    CNTV_CVAL_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHV_TVAL_EL2, Counter-timer Virtual Timer TimerValue Register (EL2)

The CNTHV_TVAL_EL2 characteristics are:

Purpose

Holds the timer value for the EL2 virtual timer.

Configuration

AArch64 System register CNTHV_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register [CNTHV_TVAL\[31:0\]](#).

This register is present only when FEAT_VHE is implemented and (EL3 is implemented or (EL3 is not implemented and FEAT_SEL2 is not implemented)). Otherwise, direct accesses to CNTHV_TVAL_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHV_TVAL_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TimerValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If [CNTHV_CTL_EL2.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTHV_CTL_EL2.ENABLE](#) is 1, the value returned is ([CNTHV_CVAL_EL2](#) - [CNTVCT_EL0](#)).

On a write of this register, [CNTHV_CVAL_EL2](#) is set to ([CNTVCT_EL0](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTHV_CTL_EL2.ENABLE](#) is 1, the timer condition is met when ([CNTVCT_EL0](#) - [CNTHV_CVAL_EL2](#)) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTHV_CTL_EL2.ISTATUS](#) is set to 1.
- If [CNTHV_CTL_EL2.IMASK](#) is 0, an interrupt is generated.

When [CNTHV_CTL_EL2.ENABLE](#) is 0, the timer condition is not met, but [CNTVCT_EL0](#) continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHV_TVAL_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic `CNTHV_TVAL_EL2` or `CNTV_TVAL_EL0` are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHV_TVAL_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if CNTHV_CTL_EL2.ENABLE == '0' then
        return bits(64) UNKNOWN;
    else
        return CNTHV_CVAL_EL2 - PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if CNTHV_CTL_EL2.ENABLE == '0' then
        return bits(64) UNKNOWN;
    else
        return CNTHV_CVAL_EL2 - PhysicalCountInt();

```

MSR CNTHV_TVAL_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHV_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    CNTHV_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();

```

MRS <Xt>, CNTV_TVAL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHVS_CVAL_EL2 - PhysicalCountInt();
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHV_CVAL_EL2 - PhysicalCountInt();
    elseif HaveEL(EL2) && (!EL2Enabled() || HCR_EL2.<E2H,TGE> != '11') then
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTV_CVAL_EL0 - PhysicalCountInt();
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif HaveEL(EL2) then
            if CNTV_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2);
        else
            if CNTV_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTV_CVAL_EL0 - PhysicalCountInt();
    elseif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
            if CNTHVS_CTL_EL2.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTHVS_CVAL_EL2 - PhysicalCountInt();
        elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
            if CNTHV_CTL_EL2.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTHV_CVAL_EL2 - PhysicalCountInt();
        elseif HCR_EL2.E2H == '0' then
            if CNTV_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2);
        else
            if CNTV_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTV_CVAL_EL0 - PhysicalCountInt();
    elseif PSTATE.EL == EL3 then
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        elseif HaveEL(EL2) && !ELUsingAArch32(EL2) then
            return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2);

```

```

elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
    return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTV0FF);
else
    return CNTV_CVAL_EL0 - PhysicalCountInt();

```

MSR CNTV_TVAL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
        IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif HaveEL(EL2) && (!EL2Enabled() || HCR_EL2.<E2H,TGE> != '11') then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL2) then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif HCR_EL2.E2H == '0' then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL3 then
    if HaveEL(EL2) && !ELUsingAArch32(EL2) then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF;
    else
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();

```

CNTHVS_CTL_EL2, Counter-timer Secure Virtual Timer Control register (EL2)

The CNTHVS CTL EL2 characteristics are:

Purpose

Control register for the Secure EL2 virtual timer.

Configuration

AArch64 System register CNTHVS_CTL_EL2 bits [31:0] are architecturally mapped to AArch32 System register [CNTHVS_CTL\[31:0\]](#).

This register is present only when FEAT_SEL2 is implemented and FEAT_VHE is implemented. Otherwise, direct accesses to CNTHVS_CTL_EL2 are UNDEFINED.

Attributes

CNTHVS CTL EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|--|--|--|--|---------|--|--|--|-------|--|--------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | RES0 | | | | | | | | ISTATUS | | | | IMASK | | ENABLE | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | |

Bits [63:3]

Reserved. RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the CNTHVS_CTL_EL2.ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the CNTHVS_CTL_EL2.ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTHVS_TVAL_EL2](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHVS_CTL_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHVS_CTL_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        return CNTHVS_CTL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        return CNTHVS_CTL_EL2;

```

MSR CNTHVS_CTL_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0100 | 0b001 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        CNTHVS_CTL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHVS_CTL_EL2 = X[t];

```

MRS <Xt>, CNTV_CTL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CTL_EL2;
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x170];
    else
        return CNTV_CTL_EL0;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CTL_EL2;
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL_EL0;
elsif PSTATE.EL == EL3 then
    return CNTV_CTL_EL0;

```

MSR CNTV_CTL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b001 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t];
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_CTL_EL2 = X[t];
    else
        CNTV_CTL_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x170] = X[t];
    else
        CNTV_CTL_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t];
    elsif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHV_CTL_EL2 = X[t];
    else
        CNTV_CTL_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    CNTV_CTL_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHVS_CVAL_EL2, Counter-timer Secure Virtual Timer CompareValue register (EL2)

The CNTHVS_CVAL_EL2 characteristics are:

Purpose

Holds the compare value for the Secure EL2 virtual timer.

Configuration

AArch64 System register CNTHVS_CVAL_EL2 bits [63:0] are architecturally mapped to AArch32 System register [CNTHVS_CVAL\[63:0\]](#).

This register is present only when FEAT_SEL2 is implemented and FEAT_VHE is implemented. Otherwise, direct accesses to CNTHVS_CVAL_EL2 are UNDEFINED.

Attributes

CNTHVS_CVAL_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

CompareValue, bits [63:0]

Holds the Secure EL2 virtual timer CompareValue.

When [CNTHVS_CTL_EL2.ENABLE](#) is 1, the timer condition is met when ([CNTVCT_EL0](#) - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTHVS_CTL_EL2.ISTATUS](#) is set to 1.
- If [CNTHVS_CTL_EL2.IMASK](#) is 0, an interrupt is generated.

When [CNTHVS_CTL_EL2.ENABLE](#) is 0, the timer condition is not met, but [CNTVCT_EL0](#) continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHVS_CVAL_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHVS_CVAL_EL2

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0100 | 0b010 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        return CNTHVS_CVAL_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        return CNTHVS_CVAL_EL2;

```

MSR CNTHVS_CVAL_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0100 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        CNTHVS_CVAL_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHVS_CVAL_EL2 = X[t];

```

MRS <Xt>, CNTV_CVAL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CVAL_EL2;
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL_EL0;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x168];
    else
        return CNTV_CVAL_EL0;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CVAL_EL2;
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL_EL0;
elseif PSTATE.EL == EL3 then
    return CNTV_CVAL_EL0;

```

MSR CNTV_CVAL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t];
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x168] = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL3 then
    CNTV_CVAL_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHVS_TVAL_EL2, Counter-timer Secure Virtual Timer TimerValue register (EL2)

The CNTHVS_TVAL_EL2 characteristics are:

Purpose

Holds the timer value for the Secure EL2 virtual timer.

Configuration

AArch64 System register CNTHVS_TVAL_EL2 bits [31:0] are architecturally mapped to AArch32 System register [CNTHVS_TVAL\[31:0\]](#).

This register is present only when FEAT_SEL2 is implemented and FEAT_VHE is implemented. Otherwise, direct accesses to CNTHVS_TVAL_EL2 are UNDEFINED.

Attributes

CNTHVS_TVAL_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TimerValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If [CNTHVS_CTL_EL2.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTHVS_CTL_EL2.ENABLE](#) is 1, the value returned is ([CNTHVS_CVAL_EL2](#) - [CNTVCT_EL0](#)).

On a write of this register, [CNTHVS_CVAL_EL2](#) is set to ([CNTVCT_EL0](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTHVS_CTL_EL2.ENABLE](#) is 1, the timer condition is met when (([CNTVCT_EL0](#) - [CNTHVS_CVAL_EL2](#)) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTHVS_CTL_EL2.ISTATUS](#) is set to 1.
- If [CNTHVS_CTL_EL2.IMASK](#) is 0, an interrupt is generated.

When [CNTHVS_CTL_EL2.ENABLE](#) is 0, the timer condition is not met, but [CNTVCT_EL0](#) continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHVS_TVAL_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTHVS_TVAL_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHVS_CVAL_EL2 - PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHVS_CVAL_EL2 - PhysicalCountInt();

```

MSR CNTHVS_TVAL_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        CNTHVS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        CNTHVS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();

```

MRS <Xt>, CNTV_TVAL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHVS_CVAL_EL2 - PhysicalCountInt();
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHV_CVAL_EL2 - PhysicalCountInt();
    elseif HaveEL(EL2) && (!EL2Enabled() || HCR_EL2.<E2H,TGE> != '11') then
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTV_CVAL_EL0 - PhysicalCountInt();
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif HaveEL(EL2) then
            if CNTV_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2);
        else
            if CNTV_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTV_CVAL_EL0 - PhysicalCountInt();
    elseif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
            if CNTHVS_CTL_EL2.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTHVS_CVAL_EL2 - PhysicalCountInt();
        elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
            if CNTHV_CTL_EL2.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTHV_CVAL_EL2 - PhysicalCountInt();
        elseif HCR_EL2.E2H == '0' then
            if CNTV_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2);
        else
            if CNTV_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTV_CVAL_EL0 - PhysicalCountInt();
    elseif PSTATE.EL == EL3 then
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        elseif HaveEL(EL2) && !ELUsingAArch32(EL2) then
            return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2);

```

```

elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
    return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTV0FF);
else
    return CNTV_CVAL_EL0 - PhysicalCountInt();

```

MSR CNTV_TVAL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
        IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif HaveEL(EL2) && (!EL2Enabled() || HCR_EL2.<E2H,TGE> != '11') then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL2) then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif HCR_EL2.E2H == '0' then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL3 then
    if HaveEL(EL2) && !ELUsingAArch32(EL2) then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF;
    else
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();

```

CNTKCTL_EL1, Counter-timer Kernel Control register

The CNTKCTL_EL1 characteristics are:

Purpose

When FEAT_VHE is not implemented, or when [HCR_EL2](#).{E2H, TGE} is not {1, 1}, this register controls the generation of an event stream from the virtual counter, and access from EL0 to the physical counter, virtual counter, EL1 physical timers, and the virtual timer.

When FEAT_VHE is implemented and [HCR_EL2](#).{E2H, TGE} is {1, 1}, this register does not cause any event stream from the virtual counter to be generated, and does not control access to the counters and timers. The access to counters and timers at EL0 is controlled by [CNTHCTL_EL2](#).

Configuration

AArch64 System register CNTKCTL_EL1 bits [31:0] are architecturally mapped to AArch32 System register [CNTKCTL\[31:0\]](#).

Attributes

CNTKCTL_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | |
|------------------------------|----|----------------|------|----|----------|---------|-------|---------|--------|----------|----------|---|
| 6362616059585756555453525150 | 49 | 48474645444342 | 41 | 40 | 39383736 | 35 | 34 | 33 | 32 | | | |
| RES0 | | | | | | | | | | | | |
| RES0 | | EVNTIS | RES0 | | ELOPTEN | ELOVTEN | EVNTI | EVNTDIR | EVNTEN | ELOVCTEN | ELOPCTEN | |
| 3130292827262524232221201918 | 17 | 16151413121110 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:18]

Reserved, RES0.

EVENTIS, bit [17]

When FEAT_ECV is implemented:

Controls the scale of the generation of the event stream.

| EVENTIS | Meaning |
|---------|---|
| 0b0 | The CNTKCTL_EL1.EVNTI field applies to CNTVCT_ELO [15:0]. |
| 0b1 | The CNTKCTL_EL1.EVNTI field applies to CNTVCT_ELO [23:8]. |

This control applies regardless of the value of the [CNTHCTL_EL2](#).ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:10]

Reserved, RES0.

ELOPTEN, bit [9]

Traps EL0 accesses to the physical timer registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2.TGE](#) is 1, as follows:

- In AArch64 state, the following registers are trapped, reported using EC syndrome value 0x18:
 - [CNTP_CTL_EL0](#), [CNTP_CVAL_EL0](#), and [CNTP_TVAL_EL0](#).
- In AArch32 state, MRC and MCR accesses to the following registers are trapped, reported using EC syndrome value 0x03, MRRC and MCRR accesses are trapped, reported using EC syndrome value 0x04:
 - [CNTP_CTL](#), [CNTP_CVAL](#), [CNTP_TVAL](#).

| ELOPTEN | Meaning |
|---------|--|
| 0b0 | EL0 accesses to the physical timer registers are trapped to EL1. |
| 0b1 | This control does not cause any instructions to be trapped. |

When FEAT_VHE is implemented and [HCR_EL2](#).{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ELOVTEN, bit [8]

Traps EL0 accesses to the virtual timer registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2.TGE](#) is 1, as follows:

- In AArch64 state, accesses to the following registers are trapped, reported using EC syndrome value 0x18:
 - [CNTV_CTL_EL0](#), [CNTV_CVAL_EL0](#), and [CNTV_TVAL_EL0](#).
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value 0x03, MRRC and MCRR accesses are trapped using EC syndrome value 0x04:
 - [CNTV_CTL](#), [CNTV_CVAL](#), and [CNTV_TVAL](#).

| ELOVTEN | Meaning |
|---------|---|
| 0b0 | EL0 accesses to the virtual timer registers are trapped. |
| 0b1 | This control does not cause any instructions to be trapped. |

When FEAT_VHE is implemented and [HCR_EL2](#).{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTI, bits [7:4]

Selects which bit of [CNTVCT_EL0](#), as seen from EL1, is the trigger for the event stream generated from that counter when that stream is enabled.

If FEAT_ECV is implemented, and CNTKCTL_EL1.EVNTIS is 1, this field selects a trigger bit in the range 8 to 23 of [CNTVCT_EL0](#).

Otherwise, this field selects a trigger bit in the range 0 to 15 of [CNTVCT_EL0](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the [CNTVCT_EL0](#) trigger bit, as seen from EL1 and defined by EVNTI, generates an event when the event stream is enabled.

| EVNTDIR | Meaning |
|---------|---|
| 0b0 | A 0 to 1 transition of the trigger bit triggers an event. |
| 0b1 | A 1 to 0 transition of the trigger bit triggers an event. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

When FEAT_VHE is not implemented, or when [HCR_EL2](#).{E2H, TGE} is not {1, 1}, enables the generation of an event stream from [CNTVCT_EL0](#) as seen from EL1.

| EVNTEN | Meaning |
|--------|----------------------------|
| 0b0 | Disables the event stream. |
| 0b1 | Enables the event stream. |

When FEAT_VHE is implemented and [HCR_EL2](#).{E2H, TGE} is {1, 1}, this control does not enable the event stream.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ELOVCTEN, bit [1]

Traps EL0 accesses to the frequency register and virtual counter register to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2](#).TGE is 1, as follows:

- In AArch64 state, accesses to the following registers are trapped and reported using EC syndrome value 0x18:
 - [CNTVCT_EL0](#) and if [CNTKCTL_EL1](#).ELOPCTEN is 0, [CNTFRQ_EL0](#).
- In AArch32 state, MRC and MCR accesses to the following registers are trapped and reported using EC syndrome value 0x03, MRRC and MCRR accesses are trapped and reported using EC syndrome value 0x04:
 - [CNTVCT](#) and if [CNTKCTL_EL1](#).ELOPCTEN is 0, [CNTFRQ](#).

| ELOVCTEN | Meaning |
|----------|---|
| 0b0 | EL0 accesses to the frequency register and virtual counter registers are trapped. |
| 0b1 | This control does not cause any instructions to be trapped. |

When FEAT_VHE is implemented and [HCR_EL2](#).{E2H, TGE} is {1, 1}, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ELOPCTEN, bit [0]

Traps EL0 accesses to the frequency register and physical counter register to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2](#).TGE is 1, as follows:

- In AArch64 state, the following registers are trapped, reported using EC syndrome value 0x18:
 - [CNTPCT_EL0](#) and if [CNTKCTL_EL1](#).ELOVCTEN is 0, [CNTFRQ_EL0](#).

- In AArch32 state, MCR or MRC accesses the following registers are trapped, reported using EC syndrome value 0x03, MCRR or MRRC accesses are trapped and reported using EC syndrome value 0x04:
 - [CNTPCT](#) and if [CNTKCTL_EL1.EL0VCTEN](#) is 0, [CNTFRQ](#).

| EL0PCTEN | Meaning |
|----------|---|
| 0b0 | EL0 accesses to the frequency register and physical counter register are trapped. |
| 0b1 | This control does not cause any instructions to be trapped. |

When FEAT_VHE is implemented and [HCR_EL2.{E2H, TGE}](#) is {1, 1}, this control does not cause any instructions to be trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTKCTL_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic CNTKCTL_EL1 or CNTKCTL_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTKCTL_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1110 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    return CNTKCTL_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CNTHCTL_EL2;
    else
        return CNTKCTL_EL1;
elsif PSTATE.EL == EL3 then
    return CNTKCTL_EL1;

```

MSR CNTKCTL_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1110 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    CNTKCTL_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CNTHCTL_EL2 = X[t];
    else
        CNTKCTL_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    CNTKCTL_EL1 = X[t];

```

MRS <Xt>, CNTKCTL_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CNTKCTL_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return CNTKCTL_EL1;
    else
        UNDEFINED;

```

MSR CNTKCTL_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CNTKCTL_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        CNTKCTL_EL1 = X[t];
    else
        UNDEFINED;

```

CNTP_CTL_EL0, Counter-timer Physical Timer Control register

The CNTP_CTL_EL0 characteristics are:

Purpose

Control register for the EL1 physical timer.

Configuration

AArch64 System register CNTP_CTL_EL0 bits [31:0] are architecturally mapped to AArch32 System register [CNTP_CTL\[31:0\]](#).

Attributes

CNTP_CTL_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTP_TVAL_EL0](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_CTL_EL0

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic CNTP_CTL_EL0 or CNTP_CTL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTP_CTL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CTL_EL2;
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL_EL0;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x180];
    else
        return CNTP_CTL_EL0;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CTL_EL2;
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL_EL0;
elseif PSTATE.EL == EL3 then
    return CNTP_CTL_EL0;

```

MSR CNTP_CTL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = X[t];
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHP_CTL_EL2 = X[t];
    else
        CNTP_CTL_EL0 = X[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x180] = X[t];
    else
        CNTP_CTL_EL0 = X[t];
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_CTL_EL2 = X[t];
    else
        CNTP_CTL_EL0 = X[t];
elseif PSTATE.EL == EL3 then
    CNTP_CTL_EL0 = X[t];

```

MRS <Xt>, CNTP_CTL_EL02

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return NVMem[0x180];
    elseif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CNTP_CTL_EL0;
    else
        UNDEFINED;
elseif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return CNTP_CTL_EL0;
    else
        UNDEFINED;

```

MSR CNTP_CTL_EL02, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x180] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CNTP_CTL_EL0 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        CNTP_CTL_EL0 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_CVAL_EL0, Counter-timer Physical Timer CompareValue register

The CNTP_CVAL_EL0 characteristics are:

Purpose

Holds the compare value for the EL1 physical timer.

Configuration

AArch64 System register CNTP_CVAL_EL0 bits [63:0] are architecturally mapped to AArch32 System register [CNTP_CVAL\[63:0\]](#).

Attributes

CNTP_CVAL_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

CompareValue, bits [63:0]

Holds the EL1 physical timer CompareValue.

When [CNTP_CTL_EL0.ENABLE](#) is 1, the timer condition is met when ([CNTPCT_EL0](#) - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTP_CTL_EL0.ISTATUS](#) is set to 1.
- If [CNTP_CTL_EL0.IMASK](#) is 0, an interrupt is generated.

When [CNTP_CTL_EL0.ENABLE](#) is 0, the timer condition is not met, but [CNTPCT_EL0](#) continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_CVAL_EL0

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic CNTP_CVAL_EL0 or CNTP_CVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTP_CVAL_EL0

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b010 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CVAL_EL2;
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHP_CVAL_EL2;
    else
        return CNTP_CVAL_EL0;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x178];
    else
        return CNTP_CVAL_EL0;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CVAL_EL2;
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHP_CVAL_EL2;
    else
        return CNTP_CVAL_EL0;
elseif PSTATE.EL == EL3 then
    return CNTP_CVAL_EL0;

```

MSR CNTP_CVAL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = X[t];
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x178] = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = X[t];
    else
        CNTP_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = X[t];

```

MRS <Xt>, CNTP_CVAL_EL02

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return NVMem[0x178];
    elseif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CNTP_CVAL_EL0;
    else
        UNDEFINED;
elseif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return CNTP_CVAL_EL0;
    else
        UNDEFINED;

```

MSR CNTP_CVAL_EL02, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVPCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x178] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CNTP_CVAL_EL0 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        CNTP_CVAL_EL0 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_TVAL_EL0, Counter-timer Physical Timer TimerValue register

The CNTP_TVAL_EL0 characteristics are:

Purpose

Holds the timer value for the EL1 physical timer.

Configuration

AArch64 System register CNTP_TVAL_EL0 bits [31:0] are architecturally mapped to AArch32 System register [CNTP_TVAL\[31:0\]](#).

Attributes

CNTP_TVAL_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TimerValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL1 physical timer.

On a read of this register:

- If [CNTP_CTL_EL0.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTP_CTL_EL0.ENABLE](#) is 1, the value returned is ([CNTP_CVAL_EL0](#) - [CNTPCT_EL0](#)).

On a write of this register, [CNTP_CVAL_EL0](#) is set to ([CNTPCT_EL0](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTP_CTL_EL0.ENABLE](#) is 1, the timer condition is met when ([CNTPCT_EL0](#) - [CNTP_CVAL_EL0](#)) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTP_CTL_EL0.ISTATUS](#) is set to 1.
- If [CNTP_CTL_EL0.IMASK](#) is 0, an interrupt is generated.

When [CNTP_CTL_EL0.ENABLE](#) is 0, the timer condition is not met, but [CNTPCT_EL0](#) continues to count, so the TimerValue view appears to continue to count down.

Note

The value of [CNTPCT_EL0](#) used in these calculations is the value seen at the Exception Level that the [CNTPCT_EL0](#) register is being read or written from.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_TVAL_EL0

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic CNTP_TVAL_EL0 or CNTP_TVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTP_TVAL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHPS_CVAL_EL2 - PhysicalCountInt();
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHP_CVAL_EL2 - PhysicalCountInt();
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
        if CNTP_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2);
    else
        if CNTP_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTP_CVAL_EL0 - PhysicalCountInt();
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
CNTHCTL_EL2.ECV == '1' then
            if CNTP_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTP_CVAL_EL0 - (PhysicalCountInt() - CNTPOFF_EL2);
        else
            if CNTP_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTP_CVAL_EL0 - PhysicalCountInt();
    elseif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
            if CNTHPS_CTL_EL2.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTHPS_CVAL_EL2 - PhysicalCountInt();
        elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
            if CNTHP_CTL_EL2.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTHP_CVAL_EL2 - PhysicalCountInt();
        else
            if CNTP_CTL_EL0.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTP_CVAL_EL0 - PhysicalCountInt();
    elseif PSTATE.EL == EL3 then
        if CNTP_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else

```

```
return CNTP_CVAL_EL0 - PhysicalCountInt();
```

MSR CNTP_TVAL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
CNTHCTL_EL2.ECV == '1' then
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTPOFF_EL2;
    else
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHP_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    else
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL3 then
    CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();

```

MRS <Xt>, CNTP_TVAL_EL02

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if CNTP_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTP_CVAL_EL0 - PhysicalCountInt();
        else
            UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        if CNTP_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTP_CVAL_EL0 - PhysicalCountInt();
    else
        UNDEFINED;

```

MSR CNTP_TVAL_EL02, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        CNTP_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    else
        UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNPCT_EL0, Counter-timer Physical Count register

The CNTPCT_EL0 characteristics are:

Purpose

Holds the 64-bit physical count value.

Configuration

AArch64 System register CNTPCT_EL0 bits [63:0] are architecturally mapped to AArch32 System register [CNPCTI\[63:0\]](#).

All reads to the CNTPCT_EL0 occur in program order relative to reads to [CNPCTSS_EL0](#) or CNTPCT_EL0.

Attributes

CNPCT_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Physical count value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Physical count value.

Reads of CNTPCT_EL0 from EL0 or EL1 return (PhysicalCountInt<63:0> - [CNTPOFF_EL2](#)<63:0>) if the access is not trapped, and all of the following are true:

- [CNTHCTL_EL2](#).ECV is 1.
- [HCR_EL2](#).{E2H, TGE} is not {1, 1}.

Where PhysicalCountInt<63:0> is the physical count returned when CNTPCT_EL0 is read from EL2 or EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPCT_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTPCT_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PCTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
        CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
            return PhysicalCountInt() - CNTPOFF_EL2;
        else
            return PhysicalCountInt();
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && CNTHCTL_EL2.EL1PCTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            if IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
            CNTHCTL_EL2.ECV == '1' then
                return PhysicalCountInt() - CNTPOFF_EL2;
            else
                return PhysicalCountInt();
    elseif PSTATE.EL == EL2 then
        return PhysicalCountInt();
    elseif PSTATE.EL == EL3 then
        return PhysicalCountInt();

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPCTSS_EL0, Counter-timer Self-Synchronized Physical Count register

The CNTPCTSS_EL0 characteristics are:

Purpose

Holds the self-synchronized view of the 64-bit physical count value.

Configuration

AArch64 System register CNTPCTSS_EL0 bits [63:0] are architecturally mapped to AArch32 System register [CNTPCTSS\[63:0\]](#).

This register is present only when FEAT_ECV is implemented. Otherwise, direct accesses to CNTPCTSS_EL0 are UNDEFINED.

All reads to the CNTPCTSS_EL0 occur in program order relative to reads to [CNTPCT_EL0](#) or CNTPCTSS_EL0.

This register is a self-synchronised view of the [CNTPCT_EL0](#) counter, and cannot be read speculatively.

Attributes

CNTPCTSS_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Self-synchronized physical count value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Self-synchronized physical count value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

- Self-synchronized physical count value.
- The reset behavior of this field is:
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPCTSS_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTPCTSS_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PCTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
        CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
            return PhysicalCountInt() - CNTPOFF_EL2;
        else
            return PhysicalCountInt();
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && CNTHCTL_EL2.EL1PCTEN == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            if IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
            CNTHCTL_EL2.ECV == '1' then
                return PhysicalCountInt() - CNTPOFF_EL2;
            else
                return PhysicalCountInt();
    elseif PSTATE.EL == EL2 then
        return PhysicalCountInt();
    elseif PSTATE.EL == EL3 then
        return PhysicalCountInt();

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPOFF_EL2, Counter-timer Physical Offset register

The CNTPOFF_EL2 characteristics are:

Purpose

Holds the 64-bit physical offset. This is the offset for the AArch64 physical timers and counters when Enhanced Counter Virtualization is enabled.

Configuration

This register is present only when FEAT_ECV is implemented. Otherwise, direct accesses to CNTPOFF_EL2 are UNDEFINED.

The CNTPOFF_EL2 offset applies to:

- Direct reads of the physical counter from EL0 or EL1.
- Indirect reads of the physical counter by the EL1 physical timer.

When EL2 is implemented and enabled in the current Security state, the physical counter uses a fixed physical offset of zero if any of the following are true:

- [CNTHCTL_EL2](#).ECV is 0.
- [SCR_EL3](#).ECVEn is 0.
- [HCR_EL2](#).{E2H, TGE} is {1, 1}.

Attributes

CNTPOFF_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Physical offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Physical offset.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPOFF_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTPOFF_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x1A8];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ECVEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ECVEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return CNTPOFF_EL2;
elsif PSTATE.EL == EL3 then
    return CNTPOFF_EL2;

```

MSR CNTPOFF_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x1A8] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ECVEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ECVEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        CNTPOFF_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CNTPOFF_EL2 = X[t];

```

CNTPS_CTL_EL1, Counter-timer Physical Secure Timer Control register

The CNTPS_CTL_EL1 characteristics are:

Purpose

Control register for the secure physical timer, usually accessible at EL3 but configurably accessible at EL1 in Secure state.

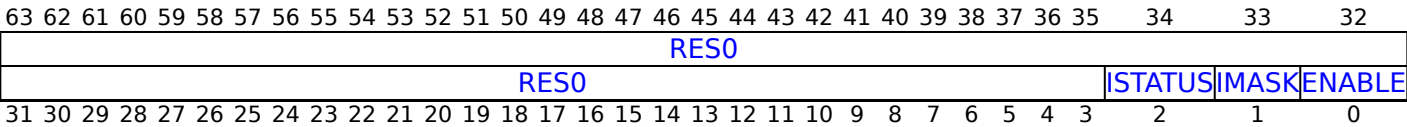
Configuration

There are no configuration notes.

Attributes

CNTPS_CTL_EL1 is a 64-bit register.

Field descriptions



Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTPS_TVAL_EL1](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPS_CTL_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTPS_CTL_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b111 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return CNTPS_CTL_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        UNDEFINED;
    elsif PSTATE.EL == EL3 then
        return CNTPS_CTL_EL1;

```

MSR CNTPS_CTL_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b111 | 0b1110 | 0b0010 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CNTPS_CTL_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    CNTPS_CTL_EL1 = X[t];
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPS_CVAL_EL1, Counter-timer Physical Secure Timer CompareValue register

The CNTPS_CVAL_EL1 characteristics are:

Purpose

Holds the compare value for the secure physical timer, usually accessible at EL3 but configurably accessible at EL1 in Secure state.

Configuration

There are no configuration notes.

Attributes

CNTPS_CVAL_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

CompareValue, bits [63:0]

Holds the secure physical timer CompareValue.

When [CNTPS_CTL_EL1.ENABLE](#) is 1, the timer condition is met when ([CNTPCT_EL0](#) - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTPS_CTL_EL1.ISTATUS](#) is set to 1.
- If [CNTPS_CTL_EL1.IMASK](#) is 0, an interrupt is generated.

When [CNTPS_CTL_EL1.ENABLE](#) is 0, the timer condition is not met, but [CNTPCT_EL0](#) continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPS_CVAL_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTPS_CVAL_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b111 | 0b1110 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elseif SCR_EL3.ST == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return CNTPS_CVAL_EL1;
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    UNDEFINED;
elseif PSTATE.EL == EL3 then
    return CNTPS_CVAL_EL1;

```

MSR CNTPS_CVAL_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b111 | 0b1110 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elseif SCR_EL3.ST == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CNTPS_CVAL_EL1 = X[t];
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    UNDEFINED;
elseif PSTATE.EL == EL3 then
    CNTPS_CVAL_EL1 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPS_TVAL_EL1, Counter-timer Physical Secure Timer TimerValue register

The CNTPS_TVAL_EL1 characteristics are:

Purpose

Holds the timer value for the secure physical timer, usually accessible at EL3 but configurably accessible at EL1 in Secure state.

Configuration

There are no configuration notes.

Attributes

CNTPS_TVAL_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | TimerValue | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the secure physical timer.

On a read of this register:

- If [CNTPS_CTL_EL1.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTPS_CTL_EL1.ENABLE](#) is 1, the value returned is ([CNTPS_CVAL_EL1](#) - [CNTPCT_EL0](#)).

On a write of this register, [CNTPS_CVAL_EL1](#) is set to ([CNTPCT_EL0](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTPS_CTL_EL1.ENABLE](#) is 1, the timer condition is met when ([CNTPCT_EL0](#) - [CNTPS_CVAL_EL1](#)) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTPS_CTL_EL1.ISTATUS](#) is set to 1.
- If [CNTPS_CTL_EL1.IMASK](#) is 0, an interrupt is generated.

When [CNTPS_CTL_EL1.ENABLE](#) is 0, the timer condition is not met, but [CNTPCT_EL0](#) continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPS_TVAL_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTPS_TVAL_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b111 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
CNTHCTL_EL2.ECV == '1' then
            if CNTPS_CTL_EL1.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTPS_CVAL_EL1 - (PhysicalCountInt() - CNTPOFF_EL2);
            else
                if CNTPS_CTL_EL1.ENABLE == '0' then
                    return bits(64) UNKNOWN;
                else
                    return CNTPS_CVAL_EL1 - PhysicalCountInt();
            else
                UNDEFINED;
        elsif PSTATE.EL == EL2 then
            UNDEFINED;
        elsif PSTATE.EL == EL3 then
            if CNTPS_CTL_EL1.ENABLE == '0' then
                return bits(64) UNKNOWN;
            else
                return CNTPS_CVAL_EL1 - PhysicalCountInt();

```

MSR CNTPS_TVAL_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b111 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && SCR_EL3.NS == '0' then
        if SCR_EL3.EEL2 == '1' then
            UNDEFINED;
        elsif SCR_EL3.ST == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && SCR_EL3.ECVEn == '1' &&
CNTHCTL_EL2.ECV == '1' then
            CNTPS_CVAL_EL1 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTPOFF_EL2;
        else
            CNTPS_CVAL_EL1 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        UNDEFINED;
    elsif PSTATE.EL == EL3 then
        CNTPS_CVAL_EL1 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();

```


CNTV_CTL_EL0, Counter-timer Virtual Timer Control register

The CNTV_CTL_EL0 characteristics are:

Purpose

Control register for the virtual timer.

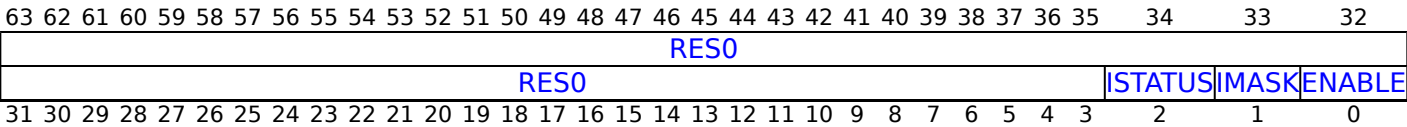
Configuration

AArch64 System register CNTV_CTL_EL0 bits [31:0] are architecturally mapped to AArch32 System register [CNTV_CTL\[31:0\]](#).

Attributes

CNTV_CTL_EL0 is a 64-bit register.

Field descriptions



Bits [63:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTV_TVAL_EL0](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_CTL_EL0

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic CNTV_CTL_EL0 or CNTV_CTL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTV_CTL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CTL_EL2;
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL_EL0;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x170];
    else
        return CNTV_CTL_EL0;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CTL_EL2;
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL_EL0;
elseif PSTATE.EL == EL3 then
    return CNTV_CTL_EL0;

```

MSR CNTV_CTL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t];
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_CTL_EL2 = X[t];
    else
        CNTV_CTL_EL0 = X[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x170] = X[t];
    else
        CNTV_CTL_EL0 = X[t];
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHV_CTL_EL2 = X[t];
    else
        CNTV_CTL_EL0 = X[t];
elseif PSTATE.EL == EL3 then
    CNTV_CTL_EL0 = X[t];

```

MRS <Xt>, CNTV_CTL_EL02

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return NVMem[0x170];
    elseif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CNTV_CTL_EL0;
    else
        UNDEFINED;
elseif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return CNTV_CTL_EL0;
    else
        UNDEFINED;

```

MSR CNTV_CTL_EL02, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x170] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CNTV_CTL_EL0 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        CNTV_CTL_EL0 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_CVAL_EL0, Counter-timer Virtual Timer CompareValue register

The CNTV_CVAL_EL0 characteristics are:

Purpose

Holds the compare value for the virtual timer.

Configuration

AArch64 System register CNTV_CVAL_EL0 bits [63:0] are architecturally mapped to AArch32 System register [CNTV_CVAL\[63:0\]](#).

Attributes

CNTV_CVAL_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

CompareValue, bits [63:0]

Holds the EL1 virtual timer CompareValue.

When [CNTV_CTL_EL0.ENABLE](#) is 1, the timer condition is met when ([CNTVCT_EL0](#) - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTV_CTL_EL0.ISTATUS](#) is set to 1.
- If [CNTV_CTL_EL0.IMASK](#) is 0, an interrupt is generated.

When [CNTV_CTL_EL0.ENABLE](#) is 0, the timer condition is not met, but [CNTVCT_EL0](#) continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_CVAL_EL0

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic CNTV_CVAL_EL0 or CNTV_CVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTV_CVAL_EL0

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b010 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CVAL_EL2;
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL_EL0;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x168];
    else
        return CNTV_CVAL_EL0;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CVAL_EL2;
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL_EL0;
elseif PSTATE.EL == EL3 then
    return CNTV_CVAL_EL0;

```

MSR CNTV_CVAL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t];
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x168] = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = X[t];
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = X[t];
    else
        CNTV_CVAL_EL0 = X[t];
elseif PSTATE.EL == EL3 then
    CNTV_CVAL_EL0 = X[t];

```

MRS <Xt>, CNTV_CVAL_EL02

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return NVMem[0x168];
    elseif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CNTV_CVAL_EL0;
    else
        UNDEFINED;
elseif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return CNTV_CVAL_EL0;
    else
        UNDEFINED;

```

MSR CNTV_CVAL_EL02, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1NVVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            NVMem[0x168] = X[t];
        elsif EL2Enabled() && HCR_EL2.NV == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' then
            CNTV_CVAL_EL0 = X[t];
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
            CNTV_CVAL_EL0 = X[t];
        else
            UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_TVAL_ELO, Counter-timer Virtual Timer TimerValue register

The CNTV_TVAL_ELO characteristics are:

Purpose

Holds the timer value for the EL1 virtual timer.

Configuration

AArch64 System register CNTV_TVAL_ELO bits [31:0] are architecturally mapped to AArch32 System register [CNTV_TVAL\[31:0\]](#).

Attributes

CNTV_TVAL_ELO is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TimerValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

TimerValue, bits [31:0]

The TimerValue view of the EL1 virtual timer.

On a read of this register:

- If [CNTV_CTL_ELO.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTV_CTL_ELO.ENABLE](#) is 1, the value returned is ([CNTV_CVAL_ELO](#) - [CNTVCT_ELO](#)).

On a write of this register, [CNTV_CVAL_ELO](#) is set to ([CNTVCT_ELO](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTV_CTL_ELO.ENABLE](#) is 1, the timer condition is met when ([CNTVCT_ELO](#) - [CNTV_CVAL_ELO](#)) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTV_CTL_ELO.ISTATUS](#) is set to 1.
- If [CNTV_CTL_ELO.IMASK](#) is 0, an interrupt is generated.

When [CNTV_CTL_ELO.ENABLE](#) is 0, the timer condition is not met, but [CNTVCT_ELO](#) continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_TVAL_EL0

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic CNTV_TVAL_EL0 or CNTV_TVAL_EL02 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTV_TVAL_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHVS_CVAL_EL2 - PhysicalCountInt();
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHV_CVAL_EL2 - PhysicalCountInt();
    elseif HaveEL(EL2) && (!EL2Enabled() || HCR_EL2.<E2H,TGE> != '11') then
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTV_CVAL_EL0 - PhysicalCountInt();
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL2) then
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTV_CVAL_EL0 - PhysicalCountInt();
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHVS_CVAL_EL2 - PhysicalCountInt();
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTHV_CVAL_EL2 - PhysicalCountInt();
    elseif HCR_EL2.E2H == '0' then
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2);
    else
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTV_CVAL_EL0 - PhysicalCountInt();
elseif PSTATE.EL == EL3 then
    if CNTV_CTL_EL0.ENABLE == '0' then
        return bits(64) UNKNOWN;
    elseif HaveEL(EL2) && !ELUsingAArch32(EL2) then
        return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTVOFF_EL2);

```

```

elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
    return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTV0FF);
else
    return CNTV_CVAL_EL0 - PhysicalCountInt();

```

MSR CNTV_TVAL_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' &&
        IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif HaveEL(EL2) && (!EL2Enabled() || HCR_EL2.<E2H,TGE> != '11') then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL2) then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif HCR_EL2.E2H == '1' && SCR_EL3.NS == '1' then
        CNTHV_CVAL_EL2 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
    elseif HCR_EL2.E2H == '0' then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    else
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();
elseif PSTATE.EL == EL3 then
    if HaveEL(EL2) && !ELUsingAArch32(EL2) then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF;
    else
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt();

```

MRS <Xt>, CNTV_TVAL_EL02

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTV0FF_EL2);
        else
            UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        if CNTV_CTL_EL0.ENABLE == '0' then
            return bits(64) UNKNOWN;
        else
            return CNTV_CVAL_EL0 - (PhysicalCountInt() - CNTV0FF_EL2);
    else
        UNDEFINED;

```

MSR CNTV_TVAL_EL02, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1110 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        CNTV_CVAL_EL0 = SignExtend((X[t]<31:0>),64) + PhysicalCountInt() - CNTV0FF_EL2;
    else
        UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVCT_EL0, Counter-timer Virtual Count register

The CNTVCT_EL0 characteristics are:

Purpose

Holds the 64-bit virtual count value. The virtual count value is equal to the physical count value minus the virtual offset visible in [CNTVOFF_EL2](#).

Configuration

AArch64 System register CNTVCT_EL0 bits [63:0] are architecturally mapped to AArch32 System register [CNTVCT\[63:0\]](#).

The value of this register is the same as the value of [CNTPCT_EL0](#) in the following conditions:

- When EL2 is not implemented.
- When EL2 is implemented, [HCR_EL2](#).E2H is 1, and this register is read from EL2.
- When EL2 is implemented and enabled in the current Security state, [HCR_EL2](#).{E2H, TGE} is {1, 1}, and this register is read from EL0 or EL2.

All reads to the CNTVCT_EL0 occur in program order relative to reads to [CNTVCTSS_EL0](#) or CNTVCT_EL0.

Attributes

CNTVCT_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual count value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual count value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVCT_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTVCT_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VCTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if HaveEL(EL2) && (!EL2Enabled() || HCR_EL2.<E2H,TGE> != '11') then
            return PhysicalCountInt() - CNTV0FF_EL2;
        else
            return PhysicalCountInt();
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && CNTHCTL_EL2.EL1TVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            if HaveEL(EL2) then
                return PhysicalCountInt() - CNTV0FF_EL2;
            else
                return PhysicalCountInt();
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '0' then
            return PhysicalCountInt() - CNTV0FF_EL2;
        else
            return PhysicalCountInt();
    elsif PSTATE.EL == EL3 then
        if HaveEL(EL2) && !ELUsingAArch32(EL2) then
            return PhysicalCountInt() - CNTV0FF_EL2;
        elsif HaveEL(EL2) && ELUsingAArch32(EL2) then
            return PhysicalCountInt() - CNTV0FF;
        else
            return PhysicalCountInt();

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVCTSS_EL0, Counter-timer Self-Synchronized Virtual Count register

The CNTVCTSS_EL0 characteristics are:

Purpose

Holds the 64-bit virtual count value. The virtual count value is equal to the physical count value visible in [CNTPCT_EL0](#) minus the virtual offset visible in [CNTVOFF_EL2](#).

Configuration

AArch64 System register CNTVCTSS_EL0 bits [63:0] are architecturally mapped to AArch32 System register [CNTVCTSS\[63:0\]](#).

This register is present only when FEAT_ECV is implemented. Otherwise, direct accesses to CNTVCTSS_EL0 are UNDEFINED.

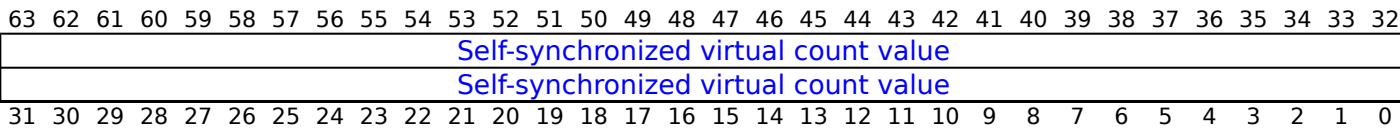
All reads to the CNTVCTSS_EL0 occur in program order relative to reads to [CNTVCT_EL0](#) or CNTVCTSS_EL0.

This register is a self-synchronised view of the [CNTVCT_EL0](#) counter, and cannot be read speculatively.

Attributes

CNTVCTSS_EL0 is a 64-bit register.

Field descriptions



Bits [63:0]

- Self-synchronized virtual count value.
- The reset behavior of this field is:
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVCTSS_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTVCTSS_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b0000 | 0b110 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VCTEN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VCTEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if HaveEL(EL2) && (!EL2Enabled() || HCR_EL2.<E2H,TGE> != '11') then
            return PhysicalCountInt() - CNTV0FF_EL2;
        else
            return PhysicalCountInt();
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && CNTHCTL_EL2.EL1TVCT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            if HaveEL(EL2) then
                return PhysicalCountInt() - CNTV0FF_EL2;
            else
                return PhysicalCountInt();
    elseif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '0' then
            return PhysicalCountInt() - CNTV0FF_EL2;
        else
            return PhysicalCountInt();
    elseif PSTATE.EL == EL3 then
        if HaveEL(EL2) && !ELUsingAArch32(EL2) then
            return PhysicalCountInt() - CNTV0FF_EL2;
        elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
            return PhysicalCountInt() - CNTV0FF;
        else
            return PhysicalCountInt();

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVOFF_EL2, Counter-timer Virtual Offset register

The CNTVOFF_EL2 characteristics are:

Purpose

Holds the 64-bit virtual offset. This is the offset between the physical count value visible in [CNTPCT_EL0](#) and the virtual count value visible in [CNTVCT_EL0](#).

Configuration

AArch64 System register CNTVOFF_EL2 bits [63:0] are architecturally mapped to AArch32 System register [CNTVOFF\[63:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3 and the virtual counter uses a fixed virtual offset of zero.

Note

When EL2 is implemented and enabled in the current Security state, and is using AArch64, the virtual counter uses a fixed virtual offset of zero in the following situations:

- [HCR_EL2.E2H](#) is 1, and [CNTVCT_EL0](#) is read from EL2.
- [HCR_EL2.{E2H, TGE}](#) is {1, 1}, and either:
 - [CNTVCT_EL0](#) is read from EL0 or EL2.
 - [CNTVCT](#) is read from EL0.

Attributes

CNTVOFF_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Virtual offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual offset.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVOFF_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CNTVOFF_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x060];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTVOFF_EL2;
elsif PSTATE.EL == EL3 then
    return CNTVOFF_EL2;

```

MSR CNTVOFF_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1110 | 0b0000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x060] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTVOFF_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CNTVOFF_EL2 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CONTEXTIDR_EL1, Context ID Register (EL1)

The CONTEXTIDR_EL1 characteristics are:

Purpose

Identifies the current Process Identifier.

The value of the whole of this register is called the Context ID and is used by:

- The debug logic, for Linked and Unlinked Context ID matching.
- The trace logic, to identify the current process.

The significance of this register is for debug and trace use only.

Configuration

AArch64 System register CONTEXTIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [CONTEXTIDR\[31:0\]](#).

Attributes

CONTEXTIDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | PROCID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

PROCID, bits [31:0]

Process Identifier. This field must be programmed with a unique value that identifies the current process.

Note

In AArch32 state, when [TTBCR.EAE](#) is set to 0, [CONTEXTIDR.ASID](#) holds the ASID.

In AArch64 state, CONTEXTIDR_EL1 is independent of the ASID, and for the EL1&0 translation regime either [TTBR0_EL1](#) or [TTBR1_EL1](#) holds the ASID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CONTEXTIDR_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic CONTEXTIDR_EL1 or CONTEXTIDR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CONTEXTIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.CONTEXTIDR_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x108];
    else
        return CONTEXTIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CONTEXTIDR_EL2;
    else
        return CONTEXTIDR_EL1;
elsif PSTATE.EL == EL3 then
    return CONTEXTIDR_EL1;

```

MSR CONTEXTIDR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.CONTEXTIDR_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x108] = X[t];
    else
        CONTEXTIDR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CONTEXTIDR_EL2 = X[t];
    else
        CONTEXTIDR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    CONTEXTIDR_EL1 = X[t];

```

MRS <Xt>, CONTEXTIDR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1101 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x108];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CONTEXTIDR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return CONTEXTIDR_EL1;
    else
        UNDEFINED;

```

MSR CONTEXTIDR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1101 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x108] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CONTEXTIDR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        CONTEXTIDR_EL1 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CONTEXTIDR_EL2, Context ID Register (EL2)

The CONTEXTIDR_EL2 characteristics are:

Purpose

Identifies the current Process Identifier for EL2.

The value of the whole of this register is called the Context ID and is used by:

- The debug logic, for Linked and Unlinked Context ID matching.
- The trace logic, to identify the current process.

The significance of this register is for debug and trace use only.

Configuration

This register is present only when FEAT_VHE is implemented or FEAT_Debugv8p2 is implemented. Otherwise, direct accesses to CONTEXTIDR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

CONTEXTIDR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | PROCID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

PROCID, bits [31:0]

Process Identifier. This field must be programmed with a unique value that identifies the current process.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CONTEXTIDR_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic CONTEXTIDR_EL2 or CONTEXTIDR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CONTEXTIDR_EL2

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1101 | 0b0000 | 0b001 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CONTEXTIDR_EL2;
elsif PSTATE.EL == EL3 then
    return CONTEXTIDR_EL2;

```

MSR CONTEXTIDR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1101 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    CONTEXTIDR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CONTEXTIDR_EL2 = X[t];

```

MRS <Xt>, CONTEXTIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.CONTEXTIDR_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x108];
    else
        return CONTEXTIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return CONTEXTIDR_EL2;
    else
        return CONTEXTIDR_EL1;
elsif PSTATE.EL == EL3 then
    return CONTEXTIDR_EL1;

```

MSR CONTEXTIDR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.CONTEXTIDR_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x108] = X[t];
    else
        CONTEXTIDR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        CONTEXTIDR_EL2 = X[t];
    else
        CONTEXTIDR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    CONTEXTIDR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPACR_EL1, Architectural Feature Access Control Register

The CPACR_EL1 characteristics are:

Purpose

Controls access to trace, SME, Streaming SVE, SVE, and Advanced SIMD and floating-point functionality.

Configuration

AArch64 System register CPACR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [CPACR\[31:0\]](#).

When EL2 is implemented and enabled in the current Security state and [HCR_EL2](#).{E2H, TGE} == {1, 1}, the fields in this register have no effect on execution at EL0 and EL1. In this case, the controls provided by [CPTR_EL2](#) are used.

Attributes

CPACR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|-----|------|----|------|----|------|----|------|----|------|----|-----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | TTA | RES0 | | SMEN | | RES0 | | FPEN | | RES0 | | ZEN | | RES0 | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:29]

Reserved, RES0.

TTA, bit [28]

Traps EL0 and EL1 System register accesses to all implemented trace registers from both Execution states to EL1, or to EL2 when it is implemented and enabled in the current Security state and [HCR_EL2](#).TGE is 1, as follows:

- In AArch64 state, accesses to trace registers are trapped, reported using ESR_ELx.EC value 0x18.
- In AArch32 state, MRC and MCR accesses to trace registers are trapped, reported using ESR_ELx.EC value 0x05.
- In AArch32 state, MRRC and MCRR accesses to trace registers are trapped, reported using ESR_ELx.EC value 0x0C.

| TTA | Meaning |
|-----|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | This control causes EL0 and EL1 System register accesses to all implemented trace registers to be trapped. |

Note

- The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the PE trace unit implements FEAT_ETMv4 or FEAT_ETE, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of [CPACR_EL1](#).TTA is 1.
- The Arm architecture does not provide traps on trace register accesses through the optional memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

If System register access to the trace functionality is not implemented, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

SMEN, bits [25:24]

When FEAT_SME is implemented:

Traps execution at EL1 and EL0 of SME instructions, SVE instructions when FEAT_SVE is not implemented or the PE is in Streaming SVE mode, and instructions that directly access the [SVCR](#) or [SMCR_EL1](#) System registers to EL1, or to EL2 when EL2 is implemented and enabled in the current Security state and [HCR_EL2](#).TGE is 1.

When instructions that directly access the [SVCR](#) System register are trapped with reference to this control, the MSR_SVCRSM, MSR_SVCRZA, and MSR_SVCRSMZA instructions are also trapped.

The exception is reported using ESR_ELx.EC value of 0x1D, with an ISS code of 0x0000000.

This field does not affect whether Streaming SVE or SME register values are valid.

A trap taken as a result of CPACR_EL1.SMEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

| SMEN | Meaning |
|------|--|
| 0b00 | This control causes execution of these instructions at EL1 and EL0 to be trapped. |
| 0b01 | This control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL1 to be trapped. |
| 0b10 | This control causes execution of these instructions at EL1 and EL0 to be trapped. |
| 0b11 | This control does not cause execution of any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [23:22]

Reserved, RES0.

FPEN, bits [21:20]

Traps execution at EL1 and EL0 of instructions that access the Advanced SIMD and floating-point registers from both Execution states to EL1, reported using ESR_ELx.EC value 0x07, or to EL2 reported using ESR_ELx.EC value 0x00 when EL2 is implemented and enabled in the current Security state and [HCR_EL2](#).TGE is 1, as follows:

- In AArch64 state, accesses to [FPCR](#), [FPSR](#), any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-31 registers.
- In AArch32 state, [FPSCR](#), and any of the SIMD and floating-point registers Q0-15, including their views as D0-D31 registers or S0-31 registers.

Traps execution at EL1 and EL0 of SME and SVE instructions to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and [HCR_EL2.TGE](#) is 1. The exception is reported using ESR_ELx.EC value 0x07.

A trap taken as a result of CPACR_EL1.SMEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

A trap taken as a result of CPACR_EL1.ZEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

| FPEN | Meaning |
|------|--|
| 0b00 | This control causes execution of these instructions at EL1 and EL0 to be trapped. |
| 0b01 | This control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL1 to be trapped. |
| 0b10 | This control causes execution of these instructions at EL1 and EL0 to be trapped. |
| 0b11 | This control does not cause execution of any instructions to be trapped. |

Writes to [MVFR0](#), [MVFR1](#), and [MVFR2](#) from EL1 or higher are CONSTRAINED UNPREDICTABLE and whether these accesses can be trapped by this control depends on implemented CONSTRAINED UNPREDICTABLE behavior.

Note

- Attempts to write to the FPSID count as use of the registers for accesses from EL1 or higher.
- Accesses from EL0 to [FPSID](#), [MVFR0](#), [MVFR1](#), [MVFR2](#), and [FPEXC](#) are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPACR_EL1.FPEN is not 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:18]

Reserved, RES0.

ZEN, bits [17:16]

When FEAT_SVE is implemented:

Traps execution at EL1 and EL0 of SVE instructions when the PE is not in Streaming SVE mode, and instructions that directly access the [ZCR_EL1](#) System register to EL1, or to EL2 when EL2 is implemented and enabled in the current Security state and [HCR_EL2.TGE](#) is 1.

The exception is reported using ESR_ELx.EC value 0x19.

A trap taken as a result of CPACR_EL1.ZEN has precedence over a trap taken as a result of CPACR_EL1.FPEN.

| ZEN | Meaning |
|------|--|
| 0b00 | This control causes execution of these instructions at EL1 and EL0 to be trapped. |
| 0b01 | This control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL1 to be trapped. |
| 0b10 | This control causes execution of these instructions at EL1 and EL0 to be trapped. |
| 0b11 | This control does not cause execution of any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [15:0]

Reserved, RES0.

Accessing CPACR_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic CPACR_EL1 or CPACR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CPACR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.CPACR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            return NVMem[0x100];
        else
            return CPACR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            return CPTR_EL2;
        else
            return CPACR_EL1;
    elsif PSTATE.EL == EL3 then
        return CPACR_EL1;

```

MSR CPACR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.CPACR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x100] = X[t];
    else
        CPACR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        CPTR_EL2 = X[t];
    else
        CPACR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    CPACR_EL1 = X[t];

```

MRS <Xt>, CPACR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0001 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x100];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return CPACR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return CPACR_EL1;
    else
        UNDEFINED;

```

MSR CPACR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0001 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x100] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            CPACR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        CPACR_EL1 = X[t];
    else
        UNDEFINED;

```


CPP RCTX, Cache Prefetch Prediction Restriction by Context

The CPP RCTX characteristics are:

Purpose

Cache Prefetch Prediction Restriction by Context applies to all Cache Allocation Resources that predict cache allocations based on information gathered within the target execution context or contexts.

Cache prefetch predictions determined by the actions of code in the target execution context or contexts appearing in program order before the instruction cannot influence speculative execution occurring after the instruction is complete and synchronized.

This instruction applies to all:

- Instruction caches.
- Data caches.
- TLB prefetching hardware used by the executing PE that applies to the supplied context or contexts.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of Cache Allocation Resources so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

Configuration

This instruction is present only when FEAT_SPECRES is implemented. Otherwise, direct accesses to CPP RCTX are UNDEFINED.

Attributes

CPP RCTX is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|------|----|----|----|----|----|----|----|----|-------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | GVMID | VMID | | | | | | | | | | | | | | | | |
| RES0 | | | | NSENS | EL | RES0 | | | | | | | | | GASID | ASID | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:49]

Reserved, RES0.

GVMID, bit [48]

Execution of this instruction applies to all VMIDs or a specified VMID.

| GVMID | Meaning |
|--------------|---|
| 0b0 | Applies to specified VMID for an EL0 or EL1 target execution context. |
| 0b1 | Applies to all VMIDs for an EL0 or EL1 target execution context. |

For target execution contexts other than EL0 and EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

VMID, bits [47:32]

Only applies when bit[48] is 0 and the target execution context is either:

- EL1.
- EL0 when ([HCR_EL2.E2H==0](#) or [HCR_EL2.TGE==0](#)).

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0 and ([HCR_EL2.E2H==0](#) or [HCR_EL2.TGE==0](#)), this field is treated as the current VMID.

When the instruction is executed at EL0 and ([HCR_EL2.E2H==1](#) and [HCR_EL2.TGE==1](#)), this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

If the implementation supports 16 bits of VMID, then the upper 8 bits of the VMID must be written to 0 by software when the context being affected only uses 8 bits.

Bits [31:28]

Reserved, RES0.

NSE, bit [27]

When FEAT_RME is implemented:

Together with the NS field, selects the Security state.

For a description of the values derived by evaluating NS and NSE together, see [CPP_RCTX.NS](#).

Otherwise:

Reserved, RES0.

NS, bit [26]

When FEAT_RME is implemented:

Together with the NSE field, selects the Security state. Defined values are:

| NSE | NS | Meaning |
|------------|-----------|----------------|
| 0b0 | 0b0 | Secure. |
| 0b0 | 0b1 | Non-secure. |
| 0b1 | 0b0 | Root. |
| 0b1 | 0b1 | Realm. |

Some Effective values are determined by the current Security state:

- When executed in Secure state, the Effective value of NSE is 0.
- When executed in Non-secure state, the Effective value of {NSE, NS} is {0, 1}.
- When executed in Realm state, the Effective value of {NSE, NS} is {1, 1}.

An instruction with an EL field that has a value other than 0b11 (EL3) is treated as a NOP when executed at EL3 with CPP_RCTX.{NSE, NS} == {1, 0}.

Otherwise:

Security State. Defined values are:

| NS | Meaning |
|-----|-------------------|
| 0b0 | Secure state. |
| 0b1 | Non-secure state. |

When executed in Non-secure state, the Effective value of NS is 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

| EL | Meaning |
|------|---------|
| 0b00 | EL0. |
| 0b01 | EL1. |
| 0b10 | EL2. |
| 0b11 | EL3. |

If the instruction is executed at an Exception level lower than the specified level, this instruction is treated as a NOP.

Bits [23:17]

Reserved, RES0.

GASID, bit [16]

Execution of this instruction applies to all ASIDs or a specified ASID.

| GASID | Meaning |
|-------|--|
| 0b0 | Applies to specified ASID for an EL0 target execution context. |
| 0b1 | Applies to all ASID for an EL0 target execution context. |

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field has an Effective value of 0.

ASID, bits [15:0]

Only applies for an EL0 target execution context and when bit[16] is 0.

Otherwise, this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being affected only uses 8 bits.

Executing the CPP RCTX instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

CPP RCTX, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b0011 | 0b111 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTL_EL1.EnRCTX == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.CPPRCTX == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTL_EL2.EnRCTX == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.RestrictPrediction(X[t], RestrictType_CachePrefetch);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.NV == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.CPPRCTX == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.RestrictPrediction(X[t], RestrictType_CachePrefetch);
    elsif PSTATE.EL == EL2 then
        AArch64.RestrictPrediction(X[t], RestrictType_CachePrefetch);
    elsif PSTATE.EL == EL3 then
        AArch64.RestrictPrediction(X[t], RestrictType_CachePrefetch);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPTR_EL2, Architectural Feature Trap Register (EL2)

The CPT_R_EL2 characteristics are:

Purpose

Controls trapping to EL2 of accesses to [CPACR](#), [CPACR_EL1](#), trace, Activity Monitor, SME, Streaming SVE, SVE, and Advanced SIMD and floating-point functionality.

Configuration

AArch64 System register CPT_R_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HCPTR\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

CPTR_EL2 is a 64-bit register.

Field descriptions

When FEAT_VHE is implemented and HCR_EL2.E2H == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|-----|------|-----|------|------|------|------|------|-----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TCPAC | TAM | RES0 | TTA | RES0 | SMEN | RES0 | FPEN | RES0 | ZEN | RES0 | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

In AArch64 state, traps accesses to [CPACR_EL1](#) from EL1 to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x18.

In AArch32 state, traps accesses to [CPACR](#) from EL1 to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x03.

| TCPAC | Meaning |
|-------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 accesses to CPACR_EL1 and CPACR are trapped to EL2, when EL2 is enabled in the current Security state. |

When [HCR_EL2.TGE](#) is 1, this control does not cause any instructions to be trapped.

Note

[CPACR_EL1](#) and [CPACR](#) are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TAM, bit [30]**When FEAT_AMUv1 is implemented:**

Trap Activity Monitor access. Traps EL1 and EL0 accesses to all Activity Monitor registers to EL2, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using ESR_ELx.EC value 0x18:
 - [AMUSERENR_EL0](#), [AMCFGR_EL0](#), [AMCGCR_EL0](#), [AMCNTENCLR0_EL0](#), [AMCNTENCLR1_EL0](#), [AMCNTENSET0_EL0](#), [AMCNTENSET1_EL0](#), [AMCR_EL0](#), [AMEVCNTR0<n>_EL0](#), [AMEVCNTR1<n>_EL0](#), [AMEVTYPE0<n>_EL0](#), and [AMEVTYPE1<n>_EL0](#).
- In AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2 and reported using ESR_ELx.EC value 0x03:
 - [AMUSERENR](#), [AMCFGR](#), [AMCGCR](#), [AMCNTENCLR0](#), [AMCNTENCLR1](#), [AMCNTENSET0](#), [AMCNTENSET1](#), [AMCR](#), [AMEVTYPE0<n>](#), and [AMEVTYPE1<n>](#).
- In AArch32 state, MRRC or MCRR accesses to [AMEVCNTR0<n>](#) and [AMEVCNTR1<n>](#), are trapped to EL2, reported using ESR_ELx.EC value 0x04.

| TAM | Meaning |
|-----|--|
| 0b0 | Accesses from EL1 and EL0 to Activity Monitor registers are not trapped. |
| 0b1 | Accesses from EL1 and EL0 to Activity Monitor registers are trapped to EL2, when EL2 is enabled in the current Security state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [29]

Reserved, RES0.

TTA, bit [28]

Traps System register accesses to all implemented trace registers from both Execution states to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to trace registers with op0=2, op1=1, and CRn<0b1000 are trapped to EL2, reported using EC syndrome value 0x18.
- In AArch32 state, MRC or MCR accesses to trace registers with cpnum=14, opc1=1, and CRn<0b1000 are trapped to EL2, reported using EC syndrome value 0x05.

| TTA | Meaning |
|-----|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any attempt at EL0, EL1 or EL2, to execute a System register access to an implemented trace register is trapped to EL2, when EL2 is enabled in the current Security state, unless HCR_EL2.TGE is 0 and it is trapped by CPACR.NSTRCDIS or CPACR_EL1.TTA . When HCR_EL2.TGE is 1, any attempt at EL0 or EL2 to execute a System register access to an implemented trace register is trapped to EL2, when EL2 is enabled in the current Security state. |

Note

The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the PE trace unit implements FEAT_ETMv4 or ETE, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPTR_EL2.TTA is 1.

EL2 does not provide traps on trace register accesses through the optional Memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

If System register access to the trace functionality is not supported, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

SMEN, bits [25:24]

When FEAT_SME is implemented:

Traps execution at EL2, EL1, and EL0 of SME instructions, SVE instructions when FEAT_SVE is not implemented or the PE is in Streaming SVE mode, and instructions that directly access the [SVCR](#), [SMCR_EL1](#), or [SMCR_EL2](#) System registers to EL2, when EL2 is enabled in the current Security state.

When instructions that directly access the [SVCR](#) System register are trapped with reference to this control, the MSR SVCRSM, MSR SVCRZA, and MSR SVCRSMZA instructions are also trapped.

The exception is reported using ESR_EL2.EC value of 0x1D, with an ISS code of 0x0000000.

This field does not affect whether Streaming SVE or SME register values are valid.

A trap taken as a result of CPTR_EL2.SMEN has precedence over a trap taken as a result of CPTR_EL2.FPEN.

| SMEN | Meaning |
|------|--|
| 0b00 | This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped. |
| 0b01 | When HCR_EL2.TGE is 0, this control does not cause execution of any instructions to be trapped. When HCR_EL2.TGE is 1, this control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL2 to be trapped. |
| 0b10 | This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped. |
| 0b11 | This control does not cause execution of any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [23:22]

Reserved, RES0.

FPEN, bits [21:20]

Traps execution at EL2, EL1, and EL0 of instructions that access the Advanced SIMD and floating-point registers from both Execution states to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x07.

Traps execution at EL2, EL1, and EL0 of SME and SVE instructions to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x07.

A trap taken as a result of CPTR_EL2.SMEN has precedence over a trap taken as a result of CPTR_EL2.FPEN.

A trap taken as a result of CPTR_EL2.ZEN has precedence over a trap taken as a result of CPTR_EL2.FPEN.

| FPEN | Meaning |
|------|--|
| 0b00 | This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped. |
| 0b01 | When HCR_EL2.TGE is 0, this control does not cause execution of any instructions to be trapped. When HCR_EL2.TGE is 1, this control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL2 to be trapped. |
| 0b10 | This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped. |
| 0b11 | This control does not cause execution of any instructions to be trapped. |

Writes to [MVFR0](#), [MVFR1](#), and [MVFR2](#) from EL1 or higher are CONSTRAINED UNPREDICTABLE and whether these accesses can be trapped by this control depends on implemented CONSTRAINED UNPREDICTABLE behavior.

Note

- Attempts to write to the FPSID count as use of the registers for accesses from EL1 or higher.
- Accesses from EL0 to [FPSID](#), [MVFR0](#), [MVFR1](#), [MVFR2](#), and [FPEXC](#) are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of CPTR_EL2.FPEN is not 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:18]

Reserved, RES0.

ZEN, bits [17:16]

When FEAT_SVE is implemented:

Traps execution at EL2, EL1, and EL0 of SVE instructions when the PE is not in Streaming SVE mode, and instructions that directly access the [ZCR_EL1](#) or [ZCR_EL2](#) System registers to EL2, when EL2 is enabled in the current Security state.

The exception is reported using ESR_ELx.EC value 0x19.

A trap taken as a result of CPTR_EL2.ZEN has precedence over a trap taken as a result of CPTR_EL2.FPEN.

| ZEN | Meaning |
|------|--|
| 0b00 | This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped. |
| 0b01 | When HCR_EL2.TGE is 0, this control does not cause execution of any instructions to be trapped. When HCR_EL2.TGE is 1, this control causes execution of these instructions at EL0 to be trapped, but does not cause execution of any instructions at EL2 to be trapped. |
| 0b10 | This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped. |
| 0b11 | This control does not cause execution of any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [15:0]

Reserved, RES0.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|-----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|------|----|----|----|----|----|----|------|----|-----|--|------|--|-----|--|------|--|----|--|------|--|--|--|--|--|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TCPAC | | TAM | | RES0 | | | | | | | | | | | | | | | | | TTA | | RES0 | | | | | | | RES1 | | TSM | | RES0 | | TFP | | RES1 | | TZ | | RES1 | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | | | | | | | |

This format applies in all Armv8.0 implementations.

Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

In AArch64 state, traps accesses to [CPACR_EL1](#) from EL1 to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x18.

In AArch32 state, traps accesses to [CPACR](#) from EL1 to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x03.

| TCPAC | Meaning |
|-------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 accesses to CPACR_EL1 and CPACR are trapped to EL2, when EL2 is enabled in the current Security state. |

When [HCR_EL2.TGE](#) is 1, this control does not cause any instructions to be trapped.

Note

[CPACR_EL1](#) and [CPACR](#) are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TAM, bit [30]**When FEAT_AMUv1 is implemented:**

Trap Activity Monitor access. Traps EL1 and EL0 accesses to all Activity Monitor registers to EL2, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using ESR_ELx.EC value 0x18:
 - [AMUSERENR_EL0](#), [AMCFGR_EL0](#), [AMCGCR_EL0](#), [AMCNTENCLR0_EL0](#), [AMCNTENCLR1_EL0](#), [AMCNTENSET0_EL0](#), [AMCNTENSET1_EL0](#), [AMCR_EL0](#), [AMEVCNTR0<n>_EL0](#), [AMEVCNTR1<n>_EL0](#), [AMEVTYPER0<n>_EL0](#), and [AMEVTYPER1<n>_EL0](#).
- In AArch32 state, MCR or MRC accesses to the following registers are trapped to EL2 and reported using ESR_ELx.EC value 0x03:
 - [AMUSERENR](#), [AMCFGR](#), [AMCGCR](#), [AMCNTENCLR0](#), [AMCNTENCLR1](#), [AMCNTENSET0](#), [AMCNTENSET1](#), [AMCR](#), [AMEVTYPER0<n>](#), and [AMEVTYPER1<n>](#).
- In AArch32 state, MCRR or MRRC accesses to [AMEVCNTR0<n>](#) and [AMEVCNTR1<n>](#), are trapped to EL2, reported using ESR_ELx.EC value 0x04.

| TAM | Meaning |
|-----|--|
| 0b0 | Accesses from EL1 and EL0 to Activity Monitor registers are not trapped. |
| 0b1 | Accesses from EL1 and EL0 to Activity Monitor registers are trapped to EL2, when EL2 is enabled in the current Security state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:21]

Reserved, RES0.

TTA, bit [20]

Traps System register accesses to all implemented trace registers from both Execution states to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to trace registers with op0=2, op1=1, and CRn<0b1000 are trapped to EL2, reported using EC syndrome value 0x18.
- In AArch32 state, MRC or MCR accesses to trace registers with cpnum=14, opc1=1, and CRn<0b1000 are trapped to EL2, reported using EC syndrome value 0x05.

| TTA | Meaning |
|-----|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any attempt at EL0, EL1, or EL2, to execute a System register access to an implemented trace register is trapped to EL2, when EL2 is enabled in the current Security state, unless it is trapped by CPACR .TRCDIS or CPACR_EL1 .TTA. |

Note

- The ETMv4 architecture does not permit EL0 to access the trace registers. If the PE trace unit implements FEAT_ETMv4, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than an exception that would be generated because the value of [CPTR_EL2](#).TTA is 1.
- EL2 does not provide traps on trace register accesses through the optional memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

If System register access to the trace functionality is not supported, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:14]

Reserved, RES0.

Bit [13]

Reserved, RES1.

TSM, bit [12]

When FEAT_SME is implemented:

Traps execution at EL2, EL1, and EL0 of SME instructions, SVE instructions when FEAT_SVE is not implemented or the PE is in Streaming SVE mode, and instructions that directly access the [SVCR](#), [SMCR_EL1](#), or [SMCR_EL2](#) System registers to EL2, when EL2 is enabled in the current Security state.

When instructions that directly access the [SVCR](#) System register are trapped with reference to this control, the MSR_SVCRSM, MSR_SVCRZA, and MSR_SVCRSMZA instructions are also trapped.

The exception is reported using ESR_EL2.EC value of 0x1D, with an ISS code of 0x0000000.

This field does not affect whether Streaming SVE or SME register values are valid.

A trap taken as a result of CPTR_EL2.TSM has precedence over a trap taken as a result of CPTR_EL2.TFP.

| TSM | Meaning |
|-----|---|
| 0b0 | This control does not cause execution of any instructions to be trapped. |
| 0b1 | This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bit [11]

Reserved, RES0.

TFP, bit [10]

Traps execution of instructions which access the Advanced SIMD and floating-point functionality, from both Execution states to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using ESR_ELx.EC value 0x07:
 - [FPCR](#), [FPSR](#), [FPEXC32_EL2](#), any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-31 registers.
- In AArch32 state, accesses to the following registers are trapped to EL2, reported using ESR_ELx.EC value 0x07:
 - [MVFR0](#), [MVFR1](#), [MVFR2](#), [FPSCR](#), [FPEXC](#), and any of the SIMD and floating-point registers Q0-15, including their views as D0-D31 registers or S0-31 registers. For the purposes of this trap,

the architecture defines a VMSR access to [FPSID](#) from EL1 or higher as an access to a SIMD and floating-point register. Otherwise, permitted VMSR accesses to [FPSID](#) are ignored.

Traps execution at the same Exception levels of SME and SVE instructions to EL2, when EL2 is enabled in the current Security state. The exception is reported using ESR_ELx.EC value 0x07.

A trap taken as a result of CPTR_EL2.TSM has precedence over a trap taken as a result of CPTR_EL2.TFP.

A trap taken as a result of CPTR_EL2.TZ has precedence over a trap taken as a result of CPTR_EL2.TFP.

| TFP | Meaning |
|-----|---|
| 0b0 | This control does not cause execution of any instructions to be trapped. |
| 0b1 | This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped. |

Note

[FPEXC32_EL2](#) is not accessible from EL0 using AArch64.

[FPSID](#), [MVFR0](#), [MVFR1](#), and [FPEXC](#) are not accessible from EL0 using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES1.

TZ, bit [8]

When FEAT_SVE is implemented:

Traps execution at EL2, EL1, and EL0 of SVE instructions when the PE is not in Streaming SVE mode, and instructions that directly access the [ZCR_EL2](#) or [ZCR_EL1](#) System registers to EL2, when EL2 is enabled in the current Security state.

The exception is reported using ESR_ELx.EC value 0x19.

A trap taken as a result of CPTR_EL2.TZ has precedence over a trap taken as a result of CPTR_EL2.TFP.

| TZ | Meaning |
|-----|---|
| 0b0 | This control does not cause execution of any instructions to be trapped. |
| 0b1 | This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bits [7:0]

Reserved, RES1.

Accessing CPTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CPTR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return CPTR_EL2;
elsif PSTATE.EL == EL3 then
    return CPTR_EL2;

```

MSR CPTR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        CPTR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    CPTR_EL2 = X[t];

```

MRS <Xt>, CPACR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.CPACR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x100];
    else
        return CPACR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return CPTR_EL2;
    else
        return CPACR_EL1;
elsif PSTATE.EL == EL3 then
    return CPACR_EL1;

```

MSR CPACR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && CPTR_EL2.TCPAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.CPACR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x100] = X[t];
    else
        CPACR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.TCPAC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        CPTR_EL2 = X[t];
    else
        CPACR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    CPACR_EL1 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPTR_EL3, Architectural Feature Trap Register (EL3)

The CPTR_EL3 characteristics are:

Purpose

Controls trapping to EL3 of accesses to [CPACR](#), [CPACR_EL1](#), [HCPTR](#), [CPTR_EL2](#), trace, Activity Monitor, SME, Streaming SVE, SVE, and Advanced SIMD and floating-point functionality.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to CPTR_EL3 are UNDEFINED.

Attributes

CPTR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|-----|------|----|----|----|----|----|----|----|----|----|-----|------|----|----|----|----|----|----|----|----|----|-----|------|-----|------|----|------|----|----|----|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TCPAC | TAM | RES0 | | | | | | | | | | TTA | RES0 | | | | | | | | | | ESM | RES0 | TFP | RES0 | EZ | RES0 | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | |

Bits [63:32]

Reserved, RES0.

TCPAC, bit [31]

Traps all of the following to EL3, from both Execution states and any Security state.

- EL2 accesses to [CPTR_EL2](#), reported using ESR_ELx.EC value 0x18, or [HCPTR](#), reported using ESR_ELx.EC value 0x03.
- EL2 and EL1 accesses to [CPACR_EL1](#) reported using ESR_ELx.EC value 0x18, or [CPACR](#) reported using ESR_ELx.EC value 0x03.

When CPTR_EL3.TCPAC is:

| TCPAC | Meaning |
|-------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL2 accesses to the CPTR_EL2 or HCPTR , and EL2 and EL1 accesses to the CPACR_EL1 or CPACR , are trapped to EL3, unless they are trapped by CPTR_EL2 .TCPAC. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TAM, bit [30]

When FEAT_AMUv1 is implemented:

Trap Activity Monitor access. Traps EL2, EL1, and EL0 accesses to all Activity Monitor registers to EL3.

Accesses to the Activity Monitors registers are trapped as follows:

- In AArch64 state, the following registers are trapped to EL3 and reported with ESR_ELx.EC value 0x18:

- [AMUSERENR_EL0](#), [AMCFGR_EL0](#), [AMCGCR_EL0](#), [AMCNTENCLR0_EL0](#), [AMCNTENCLR1_EL0](#), [AMCNTENSET0_EL0](#), [AMCNTENSET1_EL0](#), [AMCR_EL0](#), [AMEVCNTR0<n>_EL0](#), [AMEVCNTR1<n>_EL0](#), [AMEVTYPER0<n>_EL0](#), and [AMEVTYPER1<n>_EL0](#).
- In AArch32 state, accesses with MRC or MCR to the following registers reported with ESR_ELx.EC value 0x03:
 - [AMUSERENR](#), [AMCFGR](#), [AMCGCR](#), [AMCNTENCLR0](#), [AMCNTENCLR1](#), [AMCNTENSET0](#), [AMCNTENSET1](#), [AMCR](#), [AMEVTYPER0<n>](#), and [AMEVTYPER1<n>](#).
- In AArch32 state, accesses with MRRC or MCRR to the following registers, reported with ESR_ELx.EC value 0x04:
 - [AMEVCNTR0<n>](#), [AMEVCNTR1<n>](#).

| TAM | Meaning |
|-----|---|
| 0b0 | Accesses from EL2, EL1, and EL0 to Activity Monitor registers are not trapped. |
| 0b1 | Accesses from EL2, EL1, and EL0 to Activity Monitor registers are trapped to EL3. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:21]

Reserved, RES0.

TTA, bit [20]

Traps System register accesses. Accesses to the trace registers, from all Exception levels, any Security state, and both Execution states are trapped to EL3 as follows:

- In AArch64 state, Trace registers with op0=2, op1=1, and CRn<0b1000 are trapped to EL3 and reported using EC syndrome value 0x18.
- In AArch32 state, accesses using MCR or MRC to the Trace registers with cpnum=14, opc1=1, and CRn<0b1000 are reported using EC syndrome value 0x05.

| TTA | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any System register access to the trace registers is trapped to EL3, unless it is trapped by CPACR.TRCDIS , CPACR_EL1.TTA , or CPTR_EL2.TTA . |

If System register access to trace functionality is not supported, this bit is RES0.

Note

The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the PE trace unit implements FEAT_ETMv4 or FEAT_ETE, EL0 accesses to the trace registers are UNDEFINED, and any resulting exception is higher priority than this trap exception.

EL3 does not provide traps on trace register accesses through the Memory-mapped interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, no side-effects occur before the exception is taken, see 'Traps on instructions'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:13]

Reserved, RES0.

ESM, bit [12]

When FEAT_SME is implemented:

Traps execution of SME instructions, SVE instructions when FEAT_SVE is not implemented or the PE is in Streaming SVE mode, and instructions that directly access the [SMCR_EL1](#), [SMCR_EL2](#), [SMCR_EL3](#), [SMPRI_EL1](#), [SMPRMAP_EL2](#), or [SVCR](#) System registers, from all Exception levels and any Security state, to EL3.

When instructions that directly access the [SVCR](#) System register are trapped with reference to this control, the MSR SVCRSM, MSR SVCRZA, and MSR SVCRSMZA instructions are also trapped.

The exception is reported using ESR_EL3.EC value of 0x1D, with an ISS code of 0x0000000.

This field does not affect whether Streaming SVE or SME register values are valid.

A trap taken as a result of CPTR_EL3.ESM has precedence over a trap taken as a result of CPTR_EL3.TFP.

| ESM | Meaning |
|-----|--|
| 0b0 | This control causes execution of these instructions at all Exception levels to be trapped. |
| 0b1 | This control does not cause execution of any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [11]

Reserved, RES0.

TFP, bit [10]

Traps execution of instructions which access the Advanced SIMD and floating-point functionality, from all Exception levels, any Security state, and both Execution states, to EL3.

This includes the following registers, all reported using ESR_ELx.EC value 0x07:

- [FPCR](#), [FPSR](#), [FPFXC32_EL2](#), and any of the SIMD and floating-point registers V0-V31, including their views as D0-D31 registers or S0-S31 registers.
- [MVFR0](#), [MVFR1](#), [MVFR2](#), [FPSCR](#), [FPFXC](#), and any of the SIMD and floating-point registers Q0-Q15, including their views as D0-D31 registers or S0-S31 registers.
- VMSR accesses to [FPSID](#).

Permitted VMSR accesses to [FPSID](#) are ignored, but for the purposes of this trap the architecture defines a VMSR access to the [FPSID](#) from EL1 or higher as an access to a SIMD and floating-point register.

Traps execution at all Exception levels of SME and SVE instructions to EL3 from any Security state. The exception is reported using ESR_ELx.EC value 0x07.

A trap taken as a result of CPTR_EL3.ESM has precedence over a trap taken as a result of CPTR_EL3.TFP.

A trap taken as a result of CPTR_EL3.EZ has precedence over a trap taken as a result of CPTR_EL3.TFP.

Defined values are:

| TFP | Meaning |
|-----|--|
| 0b0 | This control does not cause execution of any instructions to be trapped. |
| 0b1 | This control causes execution of these instructions at all Exception levels to be trapped. |

Note

[FPEXC32_EL2](#) is not accessible from EL0 using AArch64.

[FPSID](#), [MVFR0](#), [MVFR1](#), and [FPEXC](#) are not accessible from EL0 using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

EZ, bit [8]

When FEAT_SVE is implemented:

Traps execution of SVE instructions when the PE is not in Streaming SVE mode, and instructions that directly access the [ZCR_EL3](#), [ZCR_EL2](#), or [ZCR_EL1](#) System registers, from all Exception levels and any Security state, to EL3.

The exception is reported using ESR_ELx.EC value 0x19.

A trap taken as a result of CPTR_EL3.EZ has precedence over a trap taken as a result of CPTR_EL3.TFP.

| EZ | Meaning |
|-----|--|
| 0b0 | This control causes execution of these instructions at all Exception levels to be trapped. |
| 0b1 | This control does not cause execution of any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [7:0]

Reserved, RES0.

Accessing CPTR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CPTR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return CPTR_EL3;

```

MSR CPTR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    CPTR_EL3 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CSSELR_EL1, Cache Size Selection Register

The CSSELR_EL1 characteristics are:

Purpose

Selects the current Cache Size ID Register, [CCSIDR_EL1](#), by specifying the required cache level and the cache type (either instruction or data cache).

Configuration

AArch64 System register CSSELR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [CSSELR\[31:0\]](#).

Attributes

CSSELR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|-------|----|-----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | TnD | | | | Level | | InD | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:5]

Reserved, RES0.

TnD, bit [4]

When FEAT_MTE2 is implemented:

Allocation Tag not Data bit.

| TnD | Meaning |
|-----|-------------------------------------|
| 0b0 | Data, Instruction or Unified cache. |
| 0b1 | Separate Allocation Tag cache. |

When CSSELR_EL1.InD == 1, this bit is RES0.

If CSSELR_EL1.Level is programmed to a cache level that is not implemented, then the value for this field on a read of CSSELR_EL1 is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Level, bits [3:1]

Cache level of required cache.

| Level | Meaning |
|-------|----------------|
| 0b000 | Level 1 cache. |
| 0b001 | Level 2 cache. |
| 0b010 | Level 3 cache. |
| 0b011 | Level 4 cache. |
| 0b100 | Level 5 cache. |
| 0b101 | Level 6 cache. |
| 0b110 | Level 7 cache. |

All other values are reserved.

If CSSELR_EL1.Level is programmed to a cache level that is not implemented, then the value for this field on a read of CSSELR_EL1 is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

InD, bit [0]

Instruction not Data bit.

| InD | Meaning |
|-----|------------------------|
| 0b0 | Data or unified cache. |
| 0b1 | Instruction cache. |

If CSSELR_EL1.Level is programmed to a cache level that is not implemented, then a read of CSSELR_EL1 is CONSTRAINED UNPREDICTABLE, and returns UNKNOWN values for CSSELR_EL1.{Level, InD}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CSSELR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CSSELR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b010 | 0b0000 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID2 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.TID4 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.CSSELR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return CSSELR_EL1;
elseif PSTATE.EL == EL2 then
    return CSSELR_EL1;
elseif PSTATE.EL == EL3 then
    return CSSELR_EL1;

```

MSR CSSELR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b010 | 0b0000 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID2 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TID4 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.CSSELR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        CSSELR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    CSSELR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    CSSELR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTR_EL0, Cache Type Register

The CTR_EL0 characteristics are:

Purpose

Provides information about the architecture of the caches.

Configuration

AArch64 System register CTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [CTR\[31:0\]](#).

Attributes

CTR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|-----|----|-----|----|----|----|-----|----|----|----|----------|----|----|----|------|----|------|----|----|----|----|----|----------|----|----|----|----------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | TminLine | | | | | | | |
| RES1 | RES0 | DIC | DC | CWG | | | | ERG | | | | DminLine | | | | L1Ip | | RES0 | | | | | | | | | | IminLine | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:38]

Reserved, RES0.

TminLine, bits [37:32]

When FEAT_MTE2 is implemented:

Tag minimum Line. Log₂ of the number of words covered by Allocation Tags in the smallest cache line of all caches which can contain Allocation tags that are controlled by the PE.

Note

- For an implementation with cache lines containing 64 bytes of data and 4 Allocation Tags, this will be $\log_2(64/4) = 4$.
- For an implementation with Allocations Tags in separate cache lines of 128 Allocation Tags per line, this will be $\log_2(128*16/4) = 9$.

Otherwise:

Reserved, RES0.

Bit [31]

Reserved, RES1.

Bit [30]

Reserved, RES0.

DIC, bit [29]

Instruction cache invalidation requirements for data to instruction coherence.

| DIC | Meaning |
|-----|---|
| 0b0 | Instruction cache invalidation to the Point of Unification is required for data to instruction coherence. |
| 0b1 | Instruction cache invalidation to the Point of Unification is not required for data to instruction coherence. |

IDC, bit [28]

Data cache clean requirements for instruction to data coherence. The meaning of this bit is:

| IDC | Meaning |
|-----|---|
| 0b0 | Data cache clean to the Point of Unification is required for instruction to data coherence, unless CLIDR_EL1.LoC == 0b000 or (CLIDR_EL1.LoUIS == 0b000 && CLIDR_EL1.LoUU == 0b000). |
| 0b1 | Data cache clean to the Point of Unification is not required for instruction to data coherence. |

CWG, bits [27:24]

Cache writeback granule. Log₂ of the number of words of the maximum size of memory that can be overwritten as a result of the eviction of a cache entry that has had a memory location in it modified.

A value of 0b0000 indicates that this register does not provide Cache writeback granule information and either:

- The architectural maximum of 512 words (2KB) must be assumed.
- The Cache writeback granule can be determined from maximum cache line size encoded in the Cache Size ID Registers.

Values greater than 0b1001 are reserved.

Arm recommends that an implementation that does not support cache write-back implements this field as 0b0001. This applies, for example, to an implementation that supports only write-through caches.

ERG, bits [23:20]

Exclusives reservation granule, and, if FEAT_TME is implemented, transactional reservation granule. Log₂ of the number of words of the maximum size of the reservation granule for the Load-Exclusive and Store-Exclusive instructions, and, if FEAT_TME is implemented, for detecting transactional conflicts.

A value of 0b0000 indicates that this register does not provide granule information and the architectural maximum of 512 words (2KB) must be assumed.

Value 0b0001 and values greater than 0b1001 are reserved.

DminLine, bits [19:16]

Log₂ of the number of words in the smallest cache line of all the data caches and unified caches that are controlled by the PE.

L1Ip, bits [15:14]

Level 1 instruction cache policy. Indicates the indexing and tagging policy for the L1 instruction cache. Possible values of this field are:

| L1Ip | Meaning | Applies when |
|------|--|--------------------------------|
| 0b00 | VMID aware Physical Index, Physical tag (VPIPT). | When FEAT_VPIPT is implemented |
| 0b01 | ASID-tagged Virtual Index, Virtual Tag (AIVIVT). | |
| 0b10 | Virtual Index, Physical Tag (VIPT). | |
| 0b11 | Physical Index, Physical Tag (PIPT). | |

The value 0b01 is reserved in Armv8.

The value 0b00 is permitted only in an implementation that includes FEAT_VPIPT, otherwise the value is reserved.

Bits [13:4]

Reserved, RES0.

IminLine, bits [3:0]

Log₂ of the number of words in the smallest cache line of all the instruction caches that are controlled by the PE.

Accessing CTR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CTR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0000 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCT == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TID2 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGTR_EL2.CTR_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCT == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return CTR_EL0;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID2 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.CTR_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return CTR_EL0;
    elsif PSTATE.EL == EL2 then
        return CTR_EL0;
    elsif PSTATE.EL == EL3 then
        return CTR_EL0;

```

CurrentEL, Current Exception Level

The CurrentEL characteristics are:

Purpose

Holds the current Exception level.

Configuration

There are no configuration notes.

Attributes

CurrentEL is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:4]

Reserved, RES0.

EL, bits [3:2]

Current Exception level.

| EL | Meaning |
|------|---------|
| 0b00 | EL0. |
| 0b01 | EL1. |
| 0b10 | EL2. |
| 0b11 | EL3. |

When the [HCR_EL2.NV](#) bit is 1, EL1 read accesses to the CurrentEL register return the value of 0b10 in this field.

This field resets to the highest implemented Exception level.

Bits [1:0]

Reserved, RES0.

Accessing CurrentEL

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, CurrentEL

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0010 | 0b010 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        return Zeros(60):'10':Zeros(2);
    else
        return Zeros(60):PSTATE.EL:Zeros(2);
elsif PSTATE.EL == EL2 then
    return Zeros(60):PSTATE.EL:Zeros(2);
elsif PSTATE.EL == EL3 then
    return Zeros(60):PSTATE.EL:Zeros(2);
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DACR32_EL2, Domain Access Control Register

The DACR32_EL2 characteristics are:

Purpose

Allows access to the AArch32 [DACR](#) register from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

AArch64 System register DACR32_EL2 bits [31:0] are architecturally mapped to AArch32 System register [DACR\[31:0\]](#).

This register is present only when EL1 is capable of using AArch32. Otherwise, direct accesses to DACR32_EL2 are UNDEFINED.

If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not RES0.

Attributes

DACR32_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

D<n>, bits [2n+1:2n], for n = 15 to 0

Domain n access permission, where n = 0 to 15. Permitted values are:

| D<n> | Meaning |
|------|--|
| 0b00 | No access. Any access to the domain generates a Domain fault. |
| 0b01 | Client. Accesses are checked against the permission bits in the translation tables. |
| 0b11 | Manager. Accesses are not checked against the permission bits in the translation tables. |

The value 0b10 is reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DACR32_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DACR32_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0011 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return DACR32_EL2;
elsif PSTATE.EL == EL3 then
    return DACR32_EL2;

```

MSR DACR32_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0011 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    DACR32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    DACR32_EL2 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DAIF, Interrupt Mask Bits

The DAIF characteristics are:

Purpose

Allows access to the interrupt mask bits.

Configuration

There are no configuration notes.

Attributes

DAIF is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | D | A | I | F | RES0 | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:10]

Reserved, RES0.

D, bit [9]

Process state D mask.

| D | Meaning |
|-----|--|
| 0b0 | Watchpoint, Breakpoint, and Software Step exceptions targeted at the current Exception level are not masked. |
| 0b1 | Watchpoint, Breakpoint, and Software Step exceptions targeted at the current Exception level are masked. |

When the target Exception level of the debug exception is higher than the current Exception level, the exception is not masked by this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

A, bit [8]

SError interrupt mask bit.

| A | Meaning |
|-----|-----------------------|
| 0b0 | Exception not masked. |
| 0b1 | Exception masked. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

I, bit [7]

IRQ mask bit.

| I | Meaning |
|----------|-----------------------|
| 0b0 | Exception not masked. |
| 0b1 | Exception masked. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

F, bit [6]

FIQ mask bit.

| F | Meaning |
|----------|-----------------------|
| 0b0 | Exception not masked. |
| 0b1 | Exception masked. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

Bits [5:0]

Reserved, RES0.

Accessing DAIF

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DAIF

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if (EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') || SCTLR_EL1.UMA == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            return Zeros(54):PSTATE.<D,A,I,F>:Zeros(6);
    elsif PSTATE.EL == EL1 then
        return Zeros(54):PSTATE.<D,A,I,F>:Zeros(6);
    elsif PSTATE.EL == EL2 then
        return Zeros(54):PSTATE.<D,A,I,F>:Zeros(6);
    elsif PSTATE.EL == EL3 then
        return Zeros(54):PSTATE.<D,A,I,F>:Zeros(6);

```

MSR DAIF, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if (EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') || SCTLR_EL1.UMA == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        PSTATE.<D,A,I,F> = X[t]<9:6>;
elsif PSTATE.EL == EL1 then
    PSTATE.<D,A,I,F> = X[t]<9:6>;
elsif PSTATE.EL == EL2 then
    PSTATE.<D,A,I,F> = X[t]<9:6>;
elsif PSTATE.EL == EL3 then
    PSTATE.<D,A,I,F> = X[t]<9:6>;

```

MSR DAIFSet, #<imm>

| op0 | op1 | CRn | op2 |
|------|-------|--------|-------|
| 0b00 | 0b011 | 0b0100 | 0b110 |

MSR DAIFClr, #<imm>

| op0 | op1 | CRn | op2 |
|------|-------|--------|-------|
| 0b00 | 0b011 | 0b0100 | 0b111 |

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGAUTHSTATUS_EL1, Debug Authentication Status register

The DBGAUTHSTATUS_EL1 characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

Configuration

AArch64 System register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGAUTHSTATUS\[31:0\]](#).

AArch64 System register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to External register [DBGAUTHSTATUS_EL1\[31:0\]](#).

Attributes

DBGAUTHSTATUS_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|------|----|------|----|----|----|----|----|----|----|-------|----|------|----|------|----|----|----|------|----|-----|----|-------|----|------|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | RTNID | | RTID | | RES0 | | | | | | | | RLNID | | RLID | | RES0 | | | | SNID | | SID | | NSNID | | NSID | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:28]

Reserved, RES0.

RTNID, bits [27:26]

Root non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RTID.

RTID, bits [25:24]

Root invasive debug.

| RTID | Meaning |
|------|---|
| 0b00 | Not implemented. |
| 0b10 | Implemented and disabled. ExternalRootInvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalRootInvasiveDebugEnabled() == TRUE. |

All other values are reserved.

If FEAT_RME is not implemented, the only permitted value is 00.

Bits [23:16]

Reserved, RES0.

RLNID, bits [15:14]

Realm non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RLID.

RLID, bits [13:12]

Realm invasive debug.

| RLID | Meaning |
|------|--|
| 0b00 | Not implemented. |
| 0b10 | Implemented and disabled. ExternalRealmInvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalRealmInvasiveDebugEnabled() == TRUE. |

All other values are reserved.

If FEAT_RME is not implemented, the only permitted value is 00.

Bits [11:8]

Reserved, RES0.

SNID, bits [7:6]

When FEAT_Debugv8p4 is implemented:

Secure non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.SID.

Otherwise:

Secure non-invasive debug.

| SNID | Meaning |
|------|---|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 1. |
| 0b10 | Implemented and disabled. ExternalSecureNoninvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalSecureNoninvasiveDebugEnabled() == TRUE. |

All other values are reserved.

SID, bits [5:4]

Secure invasive debug.

| SID | Meaning |
|------|---|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 1. |
| 0b10 | Implemented and disabled. ExternalSecureInvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalSecureInvasiveDebugEnabled() == TRUE. |

All other values are reserved.

NSNID, bits [3:2]**When FEAT_Debugv8p4 is implemented:**

Non-secure non-invasive debug.

| NSNID | Meaning |
|-------|--|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 0. |
| 0b11 | Implemented and enabled. EL3 is implemented or the Effective value of SCR_EL3.NS is 1. |

All other values are reserved.

Otherwise:

Non-secure non-invasive debug.

| NSNID | Meaning |
|-------|---|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 0. |
| 0b10 | Implemented and disabled. ExternalNoninvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalNoninvasiveDebugEnabled() == TRUE. |

All other values are reserved.

NSID, bits [1:0]

Non-secure invasive debug.

| NSID | Meaning |
|------|---|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 0. |
| 0b10 | Implemented and disabled. ExternalInvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalInvasiveDebugEnabled() == TRUE. |

All other values are reserved.

Accessing DBGAUTHSTATUS_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DBGAUTHSTATUS_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0111 | 0b1110 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.DBGAUTHSTATUS_EL1
== '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return DBGAUTHSTATUS_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return DBGAUTHSTATUS_EL1;
    elsif PSTATE.EL == EL3 then
        return DBGAUTHSTATUS_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGBCR<n>_EL1, Debug Breakpoint Control Registers, n = 0 - 15

The DBGBCR<n>_EL1 characteristics are:

Purpose

Holds control information for a breakpoint. Forms breakpoint n together with value register [DBGBVR<n>_EL1](#).

Configuration

AArch64 System register DBGBCR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGBCR<n>\[31:0\]](#).

AArch64 System register DBGBCR<n>_EL1 bits [31:0] are architecturally mapped to External register [DBGBCR<n>_EL1\[31:0\]](#).

If breakpoint n is not implemented, accesses to this register are UNDEFINED.

Attributes

DBGBCR<n>_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|------|----|------|----|----|----|----|----|----|----|-----|----|----|----|-----|----|-----|----|------|----|----|----|-----|----|----|----|------|----|-----|----|---|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | SSCE | | RES0 | | | | BT | | | | LBN | | | | SSC | | HMC | | RES0 | | | | BAS | | | | RES0 | | PMC | | E | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:30]

Reserved, RES0.

SSCE, bit [29]

When FEAT_RME is implemented:

Security State Control Extended.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [28:24]

Reserved, RES0.

BT, bits [23:20]

Breakpoint Type. Possible values are:

| BT | Meaning |
|--------|--|
| 0b0000 | Unlinked instruction address match. DBGBVR<n>_EL1 is the address of an instruction. |
| 0b0001 | As 0b0000, but linked to a Context matching breakpoint. |
| 0b0010 | Unlinked Context ID match. When FEAT_VHE is implemented, EL2 is using AArch64, and the Effective value of HCR_EL2.E2H is 1, if either the PE is executing at EL0 with HCR_EL2.TGE set to 1 or the PE is executing at EL2, then DBGBVR<n>_EL1.ContextID must match the CONTEXTIDR_EL2 value. Otherwise, DBGBVR<n>_EL1.ContextID must match the CONTEXTIDR_EL1 value |
| 0b0011 | As 0b0010, with linking enabled. |
| 0b0110 | Unlinked CONTEXTIDR_EL1 match. DBGBVR<n>_EL1.ContextID is a Context ID compared against CONTEXTIDR_EL1 . |
| 0b0111 | As 0b0110, with linking enabled. |
| 0b1000 | Unlinked VMID match. DBGBVR<n>_EL1.VMID is a VMID compared against VTTBR_EL2.VMID . |
| 0b1001 | As 0b1000, with linking enabled. |
| 0b1010 | Unlinked VMID and Context ID match. DBGBVR<n>_EL1.ContextID is a Context ID compared against CONTEXTIDR_EL1 , and DBGBVR<n>_EL1.VMID is a VMID compared against VTTBR_EL2.VMID . |
| 0b1011 | As 0b1010, with linking enabled. |
| 0b1100 | Unlinked CONTEXTIDR_EL2 match. DBGBVR<n>_EL1.ContextID2 is a Context ID compared against CONTEXTIDR_EL2 . |
| 0b1101 | As 0b1100, with linking enabled. |
| 0b1110 | Unlinked Full Context ID match. DBGBVR<n>_EL1.ContextID is compared against CONTEXTIDR_EL1 , and DBGBVR<n>_EL1.ContextID2 is compared against CONTEXTIDR_EL2 . |
| 0b1111 | As 0b1110, with linking enabled. |

All other values are reserved. Constraints on breakpoint programming mean other values are reserved under some conditions.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

For more information on the effect of programming the fields to a reserved value, see 'Reserved DBGBCR<n>_EL1.BT values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked breakpoint number. For Linked address matching breakpoints, this specifies the index of the Context-matching breakpoint linked to.

For all other breakpoint types this field is ignored and reads of the register return an UNKNOWN value.

This field is ignored when the value of DBGBCR<n>_EL1.E is 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Breakpoint debug event for breakpoint n is generated.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

For more information on the effect of programming the fields to a reserved set of values, see 'Reserved DBGBCR<n>_EL1.{SSC, HMC, PMC} values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Breakpoint debug event for breakpoint n is generated.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

For more information, see DBGBCR<n>_EL1.SSC.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [12:9]

Reserved, RES0.

BAS, bits [8:5]

When AArch32 is supported:

Byte address select. Defines which half-words an address-matching breakpoint matches, regardless of the instruction set and Execution state.

The permitted values depend on the breakpoint type.

For Address match breakpoints, the permitted values are:

| BAS | Match instruction at | Constraint for debuggers |
|--------|---|----------------------------------|
| 0b0011 | DBGBVR<n>_EL1 | Use for T32 instructions |
| 0b1100 | DBGBVR<n>_EL1 + 2 | Use for T32 instructions |
| 0b1111 | DBGBVR<n>_EL1 | Use for A64 and A32 instructions |

All other values are reserved. For more information, see 'Reserved DBGBCR<n>_EL1.BAS values'.

For more information on using the BAS field in address match breakpoints, see 'Using the BAS field in Address Match breakpoints'.

For Context matching breakpoints, this field is RES1 and ignored.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bits [4:3]

Reserved, RES0.

PMC, bits [2:1]

Privilege mode control. Determines the Exception level or levels at which a Breakpoint debug event for breakpoint n is generated.

The fields that indicate when the breakpoint can be generated are: HMC, PMC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

For more information, see DBGBCR<n>_EL1.SSC.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable breakpoint [DBGVCR<n>_EL1](#).

| E | Meaning |
|-----|----------------------|
| 0b0 | Breakpoint disabled. |
| 0b1 | Breakpoint enabled. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBCR<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DBGBCR<n>_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | n[3:0] | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.DBGBCRn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGBCR_EL1[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGBCR_EL1[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGBCR_EL1[UInt(CRm<3:0>)];

```

MSR DBGBCR<n>_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | n[3:0] | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.DBGBCRn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBCR_EL1[UInt(CRm<3:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBCR_EL1[UInt(CRm<3:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBCR_EL1[UInt(CRm<3:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGGBVR<n>_EL1, Debug Breakpoint Value Registers, n = 0 - 15

The DBGGBVR<n>_EL1 characteristics are:

Purpose

Holds a virtual address, or a VMID and/or a context ID, for use in breakpoint matching. Forms breakpoint n together with control register [DBGBCR<n>_EL1](#).

Configuration

AArch64 System register DBGGBVR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGGBVR<n>\[31:0\]](#).

AArch64 System register DBGGBVR<n>_EL1 bits [63:32] are architecturally mapped to AArch32 System register [DBGGBXVR<n>\[31:0\]](#).

AArch64 System register DBGGBVR<n>_EL1 bits [63:0] are architecturally mapped to External register [DBGGBVR<n>_EL1\[63:0\]](#).

How this register is interpreted depends on the value of [DBGBCR<n>_EL1](#).BT.

- When [DBGBCR<n>_EL1](#).BT is 0b000x, this register holds a virtual address.
- When [DBGBCR<n>_EL1](#).BT is 0b001x, 0b011x, or 0b110x, this register holds a Context ID.
- When [DBGBCR<n>_EL1](#).BT is 0b100x, this register holds a VMID.
- When [DBGBCR<n>_EL1](#).BT is 0b101x, this register holds a VMID and a Context ID.
- When [DBGBCR<n>_EL1](#).BT is 0b111x, this register holds two Context ID values.

For other values of [DBGBCR<n>_EL1](#).BT, this register is RES0.

If breakpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGGBVR<n>_EL1 is a 64-bit register.

Field descriptions

When DBGBCR<n>_EL1.BT == 0b000x:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RESS[14:4] | | | | | | | | | | | Bits[52:49] | | | | | VA[48:2] | | | | | | | | | | | | | | | | |
| VA[48:2] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | RES0 | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

RESS[14:4], bits [63:53]

Reserved, Sign extended. Software must set all bits in this field to the same value as the most significant bit of the VA field. If all bits in this field are not the same value as the most significant bit of the VA field, then all of the following apply:

- It is CONSTRAINED UNPREDICTABLE whether the PE ignores this field when comparing an address.
- If the breakpoint is not context-aware, it is IMPLEMENTATION DEFINED whether the value read back in each bit of this field is a copy of the most significant bit of the VA field or the value written.

VA[52:49], bits [52:49]

When FEAT_LVA is implemented:

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Extension to RESS[14:4]. For more information, see RESS[14:4].

VA[48:2], bits [48:2]

Bits[48:2] of the address value for comparison.

When FEAT_LVA is implemented, VA[52:49] forms the upper part of the address value. Otherwise, bits [52:49] are part of the RESS field.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT == 0b001x:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ContextID | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:32]

Reserved, RES0.

ContextID, bits [31:0]

Context ID value for comparison.

The value is compared against [CONTEXTIDR_EL2](#) when (FEAT_VHE is implemented or FEAT_Debugv8p2 is implemented), [HCR_EL2.E2H](#) is 1, and either:

- The PE is executing at EL2.
- [HCR_EL2.TGE](#) is 1, the PE is executing at EL0, and EL2 is enabled in the current Security state.

Otherwise, the value is compared against [CONTEXTIDR_EL1](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT == 0b011x:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ContextID | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:32]

Reserved, RES0.

ContextID, bits [31:0]

Context ID value for comparison against [CONTEXTIDR_EL1](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT == 0b100x and EL2 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | VMID[15:8] | | | | | | | | VMID[7:0] | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

VMID[15:8], bits [47:40]

When FEAT_VMID16 is implemented, VTCR_EL2.VS == 1 and EL2 is using AArch64:

Extension to VMID[7:0]. For more information, see DBGVBVR<n>_EL1.VMID[7:0].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID[7:0], bits [39:32]

VMID value for comparison.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- [VTCR_EL2](#).VS is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT == 0b101x and EL2 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | VMID[15:8] | | | | | | | | VMID[7:0] | | | | | | | |
| ContextID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

VMID[15:8], bits [47:40]

When FEAT_VMID16 is implemented, VTCR_EL2.VS == 1 and EL2 is using AArch64:

Extension to VMID[7:0]. For more information, see DBGBVR<n>_EL1.VMID[7:0].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID[7:0], bits [39:32]

VMID value for comparison.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- [VTCR_EL2.VS](#) is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

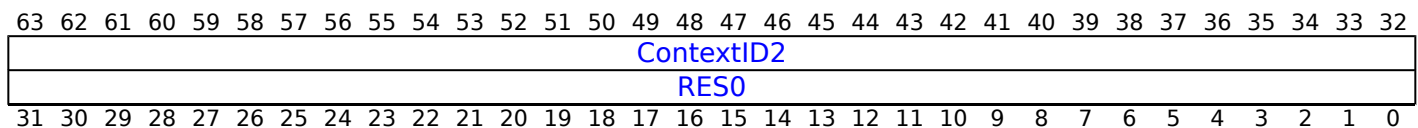
ContextID, bits [31:0]

Context ID value for comparison against [CONTEXTIDR_EL1](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT == 0b110x, EL2 is implemented and (FEAT_VHE is implemented or FEAT_Debugv8p2 is implemented):



ContextID2, bits [63:32]

Context ID value for comparison against [CONTEXTIDR_EL2](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT == 0b111x, EL2 is implemented and (FEAT_VHE is implemented or FEAT_Debugv8p2 is implemented):

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ContextID2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ContextID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ContextID2, bits [63:32]

Context ID value for comparison against [CONTEXTIDR_EL2](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

ContextID, bits [31:0]

Context ID value for comparison against [CONTEXTIDR_EL1](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBVR<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DBGBVR<n>_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | n[3:0] | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.DBGBVRn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGGBVR_EL1[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGGBVR_EL1[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGGBVR_EL1[UInt(CRm<3:0>)];

```

MSR DBGGBVR<n>_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | n[3:0] | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.DBGBVRn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBVR_EL1[UInt(CRm<3:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBVR_EL1[UInt(CRm<3:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGBVR_EL1[UInt(CRm<3:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGCLAIMCLR_EL1, Debug CLAIM Tag Clear register

The DBGCLAIMCLR_EL1 characteristics are:

Purpose

Used by software to read the values of the CLAIM tag bits, and to clear CLAIM tag bits to 0.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the [DBGCLAIMSET_EL1](#) register.

Configuration

AArch64 System register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGCLAIMCLR\[31:0\]](#).

AArch64 System register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to External register [DBGCLAIMCLR_EL1\[31:0\]](#).

An implementation must include eight CLAIM tag bits.

Attributes

DBGCLAIMCLR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|-------|----|----|----|----|--------|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | RAZ/WI | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | CLAIM | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:32]

Reserved, RES0.

Bits [31:8]

Reserved, RAZ/WI.

CLAIM, bits [7:0]

Read or clear CLAIM tag bits. Reading this field returns the current value of the CLAIM tag bits.

Writing a 1 to one of these bits clears the corresponding CLAIM tag bit to 0. This is an indirect write to the CLAIM tag bits. A single write operation can clear multiple CLAIM tag bits to 0.

Writing 0 to one of these bits has no effect.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Accessing DBGCLAIMCLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DBGCLAIMCLR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0111 | 0b1001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.DBGCLAIM == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return DBGCLAIMCLR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return DBGCLAIMCLR_EL1;
elsif PSTATE.EL == EL3 then
    return DBGCLAIMCLR_EL1;

```

MSR DBGCLAIMCLR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0111 | 0b1001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.DBGCLAIM == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGCLAIMCLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGCLAIMCLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    DBGCLAIMCLR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGCLAIMSET_EL1, Debug CLAIM Tag Set register

The DBGCLAIMSET_EL1 characteristics are:

Purpose

Used by software to set the CLAIM tag bits to 1.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the [DBGCLAIMCLR_EL1](#) register.

Configuration

AArch64 System register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGCLAIMSET\[31:0\]](#).

AArch64 System register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to External register [DBGCLAIMSET_EL1\[31:0\]](#).

An implementation must include eight CLAIM tag bits.

Attributes

DBGCLAIMSET_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:32]

Reserved, RES0.

Bits [31:8]

Reserved, RAZ/WI.

CLAIM, bits [7:0]

Set CLAIM tag bits.

This field is RAO.

Writing a 1 to one of these bits sets the corresponding CLAIM tag bit to 1. This is an indirect write to the CLAIM tag bits. A single write operation can set multiple CLAIM tag bits to 1.

Writing 0 to one of these bits has no effect.

Accessing DBGCLAIMSET_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DBGCLAIMSET_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0111 | 0b1000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.DBGCLAIM == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return DBGCLAIMSET_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return DBGCLAIMSET_EL1;
elsif PSTATE.EL == EL3 then
    return DBGCLAIMSET_EL1;

```

MSR DBGCLAIMSET_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0111 | 0b1000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.DBGCLAIM == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGCLAIMSET_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGCLAIMSET_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    DBGCLAIMSET_EL1 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTR_EL0, Debug Data Transfer Register, half-duplex

The DBGDTR_EL0 characteristics are:

Purpose

Transfers 64 bits of data between the PE and an external debugger. Can transfer both ways using only a single register.

Configuration

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch32 System register [DBGDTRRXint\[31:0\]](#) when written.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to External register [DBGDTRRX_EL0\[31:0\]](#) when written.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch64 System register [DBGDTRRX_EL0\[31:0\]](#) when written.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [DBGDTRTXint\[31:0\]](#) when written.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to External register [DBGDTRTX_EL0\[31:0\]](#) when written.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch64 System register [DBGDTRTX_EL0\[31:0\]](#) when written.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch32 System register [DBGDTRTXint\[31:0\]](#) when read.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to External register [DBGDTRTX_EL0\[31:0\]](#) when read.

AArch64 System register DBGDTR_EL0 bits [63:32] are architecturally mapped to AArch64 System register [DBGDTRTX_EL0\[31:0\]](#) when read.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [DBGDTRRXint\[31:0\]](#) when read.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to External register [DBGDTRRX_EL0\[31:0\]](#) when read.

AArch64 System register DBGDTR_EL0 bits [31:0] are architecturally mapped to AArch64 System register [DBGDTRRX_EL0\[31:0\]](#) when read.

Attributes

DBGDTR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| HighWord | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| LowWord | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

HighWord, bits [63:32]

Writes to this register set DTRRX to the value in this field and do not change RXfull.

Reads of this register:

- If RXfull is set to 1, return the last value written to DTRTX.
- If RXfull is set to 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

LowWord, bits [31:0]

Writes to this register set DTRTX to the value in this field and set TXfull to 1.

Reads of this register:

- If RXfull is set to 1, return the last value written to DTRRX.
- If RXfull is set to 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

Accessing DBGDTR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DBGDTR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b011 | 0b0000 | 0b0100 | 0b000 |

```

if Halted() then
    return DBGDTR_EL0;
elsif PSTATE.EL == EL0 then
    if MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return DBGDTR_EL0;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && MDCR_EL2.TDCC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return DBGDTR_EL0;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return DBGDTR_EL0;
    elsif PSTATE.EL == EL3 then
        return DBGDTR_EL0;

```

MSR DBGDTR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b011 | 0b0000 | 0b0100 | 0b000 |

```

if Halted() then
    DBGDTR_EL0 = X[t];
elsif PSTATE.EL == EL0 then
    if MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGDTR_EL0 = X[t];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && MDCR_EL2.TDCC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGDTR_EL0 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGDTR_EL0 = X[t];
    elsif PSTATE.EL == EL3 then
        DBGDTR_EL0 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRRX_EL0, Debug Data Transfer Register, Receive

The DBGDTRRX_EL0 characteristics are:

Purpose

Transfers data from an external debugger to the PE. For example, it is used by a debugger transferring commands and data to a debug target. See [DBGDTR_EL0](#) for additional architectural mappings. It is a component of the Debug Communications Channel.

Configuration

AArch64 System register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to AArch32 System register [DBGDTRRXint\[31:0\]](#).

AArch64 System register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to External register [DBGDTRRX_EL0\[31:0\]](#).

Attributes

DBGDTRRX_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Update DTRRX | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

Bits [31:0]

Update DTRRX.

Reads of this register:

- If RXfull is set to 1, return the last value written to DTRRX.
- If RXfull is set to 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRRX_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DBGDTRRX_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b011 | 0b0000 | 0b0101 | 0b000 |

```

if Halted() then
    return DBGDTRRX_EL0;
elsif PSTATE.EL == EL0 then
    if MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return DBGDTRRX_EL0;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && MDCR_EL2.TDCC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return DBGDTRRX_EL0;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return DBGDTRRX_EL0;
    elsif PSTATE.EL == EL3 then
        return DBGDTRRX_EL0;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRTX_EL0, Debug Data Transfer Register, Transmit

The DBGDTRTX_EL0 characteristics are:

Purpose

Transfers data from the PE to an external debugger. For example, it is used by a debug target to transfer data to the debugger. See [DBGDTR_EL0](#) for additional architectural mappings. It is a component of the Debug Communication Channel.

Configuration

AArch64 System register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to AArch32 System register [DBGDTRTXint\[31:0\]](#).

AArch64 System register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to External register [DBGDTRTX_EL0\[31:0\]](#).

Attributes

DBGDTRTX_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Return DTRTX | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

Bits [31:0]

Return DTRTX.

Writes to this register:

- If TXfull is set to 1, set DTRRX and DTRTX to UNKNOWN.
- If TXfull is set to 0, update the value in DTRTX.

After the write, TXfull is set to 1.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRTX_EL0

Accesses to this register use the following encodings in the System register encoding space:

MSR DBGDTRTX_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b011 | 0b0000 | 0b0101 | 0b000 |

```

if Halted() then
    DBGDTRTX_EL0 = X[t];
elsif PSTATE.EL == EL0 then
    if MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGDTRTX_EL0 = X[t];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && MDCR_EL2.TDCC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGDTRTX_EL0 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGDTRTX_EL0 = X[t];
    elsif PSTATE.EL == EL3 then
        DBGDTRTX_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGPRCR_EL1, Debug Power Control Register

The DBGPRCR_EL1 characteristics are:

Purpose

Controls behavior of the PE on powerdown request.

Configuration

AArch64 System register DBGPRCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGPRCR\[31:0\]](#).

Bit [0] of this register is mapped to [EDPRCR](#).CORENPDRQ, bit [0] of the external view of this register.

The other bits in these registers are not mapped to each other.

Attributes

DBGPRCR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | CORENPDRQ |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:1]

Reserved, RES0.

CORENPDRQ, bit [0]

When FEAT_DoPD is implemented:

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

| CORENPDRQ | Meaning |
|-----------|--|
| 0b0 | If the system responds to a powerdown request, it powers down Core power domain. |
| 0b1 | If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain. |

In an implementation that includes the recommended external debug interface, this bit drives the DBGNOPWRDWN signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to its Cold reset value on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states see 'Core power domain power states'.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

On a Cold reset, if the powerup request is implemented and the powerup request has been asserted, this field is set to an IMPLEMENTATION DEFINED choice of 0 or 1. If the powerup request is not asserted, this field is set to 0.

Otherwise:

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

| CORENPDRQ | Meaning |
|-----------|--|
| 0b0 | If the system responds to a powerdown request, it powers down Core power domain. |
| 0b1 | If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain. |

In an implementation that includes the recommended external debug interface, this bit drives the DBGNOPWRDWN signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to the value of [EDPRCR.COREPURQ](#) on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states see 'Core power domain power states'.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The reset behavior of this field is:

- On a Cold reset, this field resets to the value in [EDPRCR.COREPURQ](#).

Accessing DBGPRCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DBGPRCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0001 | 0b0100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.DBGPRCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return DBGPRCR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDOSA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return DBGPRCR_EL1;
    elsif PSTATE.EL == EL3 then
        return DBGPRCR_EL1;

```

MSR DBGPRCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0001 | 0b0100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.DBGPRCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGPRCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            DBGPRCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    DBGPRCR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGVCR32_EL2, Debug Vector Catch Register

The DBGVCR32_EL2 characteristics are:

Purpose

Allows access to the AArch32 register [DBGVCR](#) from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

AArch64 System register DBGVCR32_EL2 bits [31:0] are architecturally mapped to AArch32 System register [DBGVCR\[31:0\]](#).

This register is present only when EL1 is capable of using AArch32. Otherwise, direct accesses to DBGVCR32_EL2 are UNDEFINED.

If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not RES0.

Attributes

DBGVCR32_EL2 is a 64-bit register.

Field descriptions

When EL3 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|------|-----|-----|-----|-----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| NSF | NSI | RES0 | NSD | NSP | NSS | NSU | RES0 | | | | | | | | | | | | | | | | | SF | SI | RES0 | SD | SP | SS | SU | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

NSF, bit [31]

FIQ vector catch enable in Non-secure state.

The exception vector offset is 0x1C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSI, bit [30]

IRQ vector catch enable in Non-secure state.

The exception vector offset is 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [29]

Reserved, RES0.

NSD, bit [28]

Data Abort vector catch enable in Non-secure state.

The exception vector offset is 0x10.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSP, bit [27]

Prefetch Abort vector catch enable in Non-secure state.

The exception vector offset is 0x0C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSS, bit [26]

Supervisor Call (SVC) vector catch enable in Non-secure state.

The exception vector offset is 0x08.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSU, bit [25]

Undefined Instruction vector catch enable in Non-secure state.

The exception vector offset is 0x04.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [24:8]

Reserved, RES0.

SF, bit [7]

FIQ vector catch enable in Secure state.

The exception vector offset is 0x1C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SI, bit [6]

IRQ vector catch enable in Secure state.

The exception vector offset is 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

SD, bit [4]

Data Abort vector catch enable in Secure state.

The exception vector offset is 0x10.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SP, bit [3]

Prefetch Abort vector catch enable in Secure state.

The exception vector offset is 0x0C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SS, bit [2]

Supervisor Call (SVC) vector catch enable in Secure state.

The exception vector offset is 0x08.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SU, bit [1]

Undefined Instruction vector catch enable in Secure state.

The exception vector offset is 0x04.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

When EL3 is not implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|------|----|----|----|----|------|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | F | | I | RES0 | D | P | S | U | RES0 | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:8]

Reserved, RES0.

F, bit [7]

FIQ vector catch enable.

The exception vector offset is 0x1C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [6]

IRQ vector catch enable.

The exception vector offset is 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

D, bit [4]

Data Abort vector catch enable.

The exception vector offset is 0x10.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P, bit [3]

Prefetch Abort vector catch enable.

The exception vector offset 0x0C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [2]

Supervisor Call (SVC) vector catch enable.

The exception vector offset is 0x08.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

U, bit [1]

Undefined Instruction vector catch enable.

The exception vector offset is 0x04.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

Accessing DBGVCR32_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DBGVCR32_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b100 | 0b0000 | 0b0111 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return DBGVCR32_EL2;
elsif PSTATE.EL == EL3 then
    return DBGVCR32_EL2;

```

MSR DBGVCR32_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b100 | 0b0000 | 0b0111 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        DBGVCR32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    DBGVCR32_EL2 = X[t];

```


DBGWCR<n>_EL1, Debug Watchpoint Control Registers, n = 0 - 15

The DBGWCR<n>_EL1 characteristics are:

Purpose

Holds control information for a watchpoint. Forms watchpoint n together with value register [DBGWVR<n>_EL1](#).

Configuration

AArch64 System register DBGWCR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGWCR<n>\[31:0\]](#).

AArch64 System register DBGWCR<n>_EL1 bits [31:0] are architecturally mapped to External register [DBGWCR<n>_EL1\[31:0\]](#).

If watchpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGWCR<n>_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|------|----|------|----|----|----|------|----|----|----|-----|----|----|----|-----|----|-----|----|-----|----|----|----|----|----|----|----|-----|----|-----|----|---|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | SSCE | | MASK | | | | RES0 | | WT | | LBN | | | | SSC | | HMC | | BAS | | | | | | | | LSC | | PAC | | E | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:30]

Reserved, RES0.

SSCE, bit [29]

When FEAT_RME is implemented:

Security State Control Extended.

The fields that indicate when the watchpoint can be generated are: HMC, PAC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MASK, bits [28:24]

Address mask. Only objects up to 2GB can be watched using a single mask.

| MASK | Meaning |
|-------------|----------------|
| 0b00000 | No mask. |
| 0b00001 | Reserved. |
| 0b00010 | Reserved. |

If programmed with a reserved value, a watchpoint must behave as if either:

- MASK has been programmed with a defined value, which might be 0 (no mask), other than for a direct read of DBGWCRn_EL1.
- The watchpoint is disabled.

Software must not rely on this property because the behavior of reserved values might change in a future revision of the architecture.

Other values mask the corresponding number of address bits, from 0b00011 masking 3 address bits (0x00000007 mask for address) to 0b11111 masking 31 address bits (0x7FFFFFFF mask for address).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [23:21]

Reserved, RES0.

WT, bit [20]

Watchpoint type. Possible values are:

| WT | Meaning |
|-----------|------------------------------|
| 0b0 | Unlinked data address match. |
| 0b1 | Linked data address match. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked breakpoint number. For Linked data address watchpoints, this specifies the index of the Context-matching breakpoint linked to.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Watchpoint debug event for watchpoint n is generated.

The fields that indicate when the watchpoint can be generated are: HMC, PAC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

For more information on the effect of programming the fields to a reserved value, see 'Reserved DBGWCR<n>_EL1.{SSC, HMC, PAC} values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Watchpoint debug event for watchpoint n is generated.

The fields that indicate when the watchpoint can be generated are: HMC, PAC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

BAS, bits [12:5]

Byte address select. Each bit of this field selects whether a byte from within the word or double-word addressed by [DBGWVR<n>_EL1](#) is being watched.

| BAS | Description |
|----------|---|
| xxxxxxx1 | Match byte at DBGWVR<n>_EL1 |
| xxxxxx1x | Match byte at DBGWVR<n>_EL1 + 1 |
| xxxxx1xx | Match byte at DBGWVR<n>_EL1 + 2 |
| xxx1xxx | Match byte at DBGWVR<n>_EL1 + 3 |

In cases where [DBGWVR<n>_EL1](#) addresses a double-word:

| BAS | Description, if DBGWVR<n>_EL1[2] == 0 |
|----------|---|
| xxx1xxxx | Match byte at DBGWVR<n>_EL1 + 4 |
| xx1xxxxx | Match byte at DBGWVR<n>_EL1 + 5 |
| x1xxxxxx | Match byte at DBGWVR<n>_EL1 + 6 |
| 1xxxxxxx | Match byte at DBGWVR<n>_EL1 + 7 |

If [DBGWVR<n>_EL1\[2\]](#) == 1, only BAS[3:0] are used and BAS[7:4] are ignored. Arm deprecates setting [DBGWVR<n>_EL1\[2\]](#) == 1.

The valid values for BAS are non-zero binary numbers all of whose set bits are contiguous. All other values are reserved and must not be used by software. See 'Reserved DBGWCR<n>_EL1.BAS values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LSC, bits [4:3]

Load/store control. This field enables watchpoint matching on the type of access being made. Possible values of this field are:

| LSC | Meaning |
|------|---|
| 0b01 | Match instructions that load from a watchpointed address. |
| 0b10 | Match instructions that store to a watchpointed address. |
| 0b11 | Match instructions that load from or store to a watchpointed address. |

All other values are reserved, but must behave as if the watchpoint is disabled. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PAC, bits [2:1]

Privilege of access control. Determines the Exception level or levels at which a Watchpoint debug event for watchpoint n is generated.

The fields that indicate when the watchpoint can be generated are: HMC, PAC, SSC, and SSCE. These fields must be considered in combination, and the values that are permitted for these fields are constrained.

For more information on the operation of these fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable watchpoint n. Possible values are:

| E | Meaning |
|-----|----------------------|
| 0b0 | Watchpoint disabled. |
| 0b1 | Watchpoint enabled. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGWCR<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DBGWCR<n>_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | n[3:0] | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.DBGWCRn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGWCR_EL1[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGWCR_EL1[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGWCR_EL1[UInt(CRm<3:0>)];

```

MSR DBGWCR<n>_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | n[3:0] | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.DBGWCRn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWCR_EL1[UInt(CRm<3:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWCR_EL1[UInt(CRm<3:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWCR_EL1[UInt(CRm<3:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGWVR<n>_EL1, Debug Watchpoint Value Registers, n = 0 - 15

The DBGWVR<n>_EL1 characteristics are:

Purpose

Holds a data address value for use in watchpoint matching. Forms watchpoint n together with control register [DBGWCR<n>_EL1](#).

Configuration

AArch64 System register DBGWVR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGWVR<n>\[31:0\]](#).

AArch64 System register DBGWVR<n>_EL1 bits [63:0] are architecturally mapped to External register [DBGWVR<n>_EL1\[63:0\]](#).

If watchpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGWVR<n>_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RESS[14:4] | | | | | | | | | | | Bits[52:49] | | | VA[48:2] | | | | | | | | | | | | | | | | | | |
| VA[48:2] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | RES0 | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

RESS[14:4], bits [63:53]

Reserved, Sign extended. Software must set all bits in this field to the same value as the most significant bit of the VA field. If all bits in this field are not the same value as the most significant bit of the VA field, then all of the following apply:

- It is CONSTRAINED UNPREDICTABLE whether the PE ignores this field when comparing an address.
- It is IMPLEMENTATION DEFINED whether the value read back in each bit of this field is a copy of the most significant bit of the VA field or the value written.

VA[52:49], bits [52:49]

When FEAT_LVA is implemented:

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Extension to RESS[14:4]. For more information, see RESS[14:4].

VA[48:2], bits [48:2]

Bits[48:2] of the address value for comparison.

When FEAT_LVA is implemented, VA[52:49] forms the upper part of the address value. Otherwise, bits [52:49] are part of the RESS field.

Arm deprecates setting [DBGWVR<n>_EL1\[2\] == 1](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

Accessing DBGWVR<n>_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DBGWVR<n>_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | n[3:0] | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.DBGWVRn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGWVR_EL1[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGWVR_EL1[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGWVR_EL1[UInt(CRm<3:0>)];

```

MSR DBGWVR<n>_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | n[3:0] | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.DBGWVRn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWVR_EL1[UInt(CRm<3:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWVR_EL1[UInt(CRm<3:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWVR_EL1[UInt(CRm<3:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGDSW, Clean of Data and Allocation Tags by Set/Way

The DC CGDSW characteristics are:

Purpose

Clean data and Allocation Tags in data cache by set/way.

Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC CGDSW are UNDEFINED.

Attributes

DC CGDSW is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SetWay | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Level | | RES0 | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

$A = \text{Log}_2(\text{ASSOCIATIVITY})$, $L = \text{Log}_2(\text{LINELEN})$, $B = (L + S)$, $S = \text{Log}_2(\text{NSETS})$.

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing the DC CGDSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
 - No cache lines.
 - A single arbitrary cache line.
 - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CGDSW, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b1010 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_SetWay);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGDVAC, Clean of Data and Allocation Tags by VA to PoC

The DC CGDVAC characteristics are:

Purpose

Clean data and Allocation Tags in data cache by address to Point of Coherency.

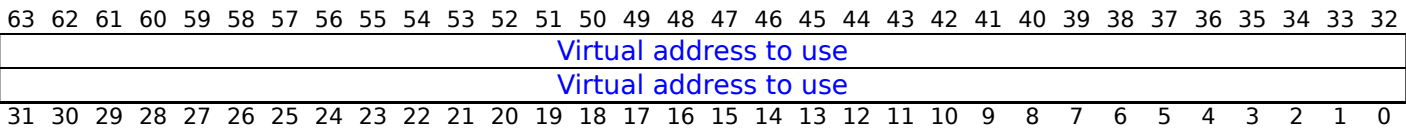
Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC CGDVAC are UNDEFINED.

Attributes

DC CGDVAC is a 64-bit System instruction.

Field descriptions



Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC CGDVAC instruction

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission fault, subject to the constraints described in 'Permission fault'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CGDVAC, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b1010 | 0b101 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCCVAC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoC);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoC);
    elsif PSTATE.EL == EL2 then
        AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoC);
    elsif PSTATE.EL == EL3 then
        AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoC);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGDVADP, Clean of Data and Allocation Tags by VA to PoDP

The DC CGDVADP characteristics are:

Purpose

Clean Allocation Tags and data in data cache by address to Point of Deep Persistence.

If the memory system does not identify a Point of Deep Persistence, then this instruction behaves as a [DC CGDVAP](#).

Configuration

This instruction is present only when FEAT_DPB2 is implemented and FEAT_MTE is implemented. Otherwise, direct accesses to DC CGDVADP are UNDEFINED.

Attributes

DC CGDVADP is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC CGDVADP instruction

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission fault, see 'Permission fault'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CGDVADP, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b1101 | 0b101 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCCVADP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoDP);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVADP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoDP);
    elsif PSTATE.EL == EL2 then
        AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoDP);
    elsif PSTATE.EL == EL3 then
        AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoDP);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGDVAP, Clean of Data and Allocation Tags by VA to PoP

The DC CGDVAP characteristics are:

Purpose

Clean data and Allocation Tags in data cache by address to Point of Persistence.

If the memory system does not identify a Point of Persistence, then this instruction behaves as a [DC CGDVAC](#).

Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC CGDVAP are UNDEFINED.

Attributes

DC CGDVAP is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC CGDVAP instruction

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission fault, see 'Permission fault'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CGDVAP, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b1100 | 0b101 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCCVAP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoP);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoP);
    elsif PSTATE.EL == EL2 then
        AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoP);
    elsif PSTATE.EL == EL3 then
        AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Clean, CacheOpScope_PoP);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGSW, Clean of Allocation Tags by Set/Way

The DC CGSW characteristics are:

Purpose

Clean Allocation Tags in data cache by set/way.

Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC CGSW are UNDEFINED.

Attributes

DC CGSW is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|-------|----|----|----|------|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| SetWay | | | | | | | | | | | | | | | | | | | | | | | | | | Level | | | | RES0 | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

$A = \text{Log}_2(\text{ASSOCIATIVITY})$, $L = \text{Log}_2(\text{LINELEN})$, $B = (L + S)$, $S = \text{Log}_2(\text{NSETS})$.

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing the DC CGSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
 - No cache lines.
 - A single arbitrary cache line.
 - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CGSW, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b1010 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_SetWay);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGVAC, Clean of Allocation Tags by VA to PoC

The DC CGVAC characteristics are:

Purpose

Clean Allocation Tags in data cache by address to Point of Coherency.

Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC CGVAC are UNDEFINED.

Attributes

DC CGVAC is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC CGVAC instruction

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission fault, subject to the constraints described in 'Permission fault'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CGVAC, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b1010 | 0b011 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCCVAC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoC);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoC);
    elsif PSTATE.EL == EL2 then
        AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoC);
    elsif PSTATE.EL == EL3 then
        AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoC);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGVADP, Clean of Allocation Tags by VA to PoDP

The DC CGVADP characteristics are:

Purpose

Clean Allocation tags by address to Point of Deep Persistence.

If the memory system does not identify a Point of Deep Persistence, then this instruction behaves as a [DC CGVAP](#).

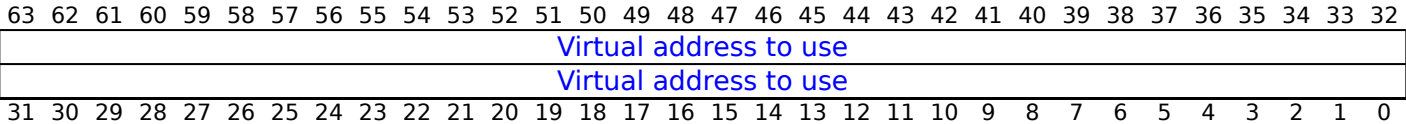
Configuration

This instruction is present only when FEAT_DPB2 is implemented and FEAT_MTE is implemented. Otherwise, direct accesses to DC CGVADP are UNDEFINED.

Attributes

DC CGVADP is a 64-bit System instruction.

Field descriptions



Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC CGVADP instruction

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission fault, see 'Permission fault'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CGVADP, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b1101 | 0b011 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCCVADP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoDP);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVADP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoDP);
    elsif PSTATE.EL == EL2 then
        AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoDP);
    elsif PSTATE.EL == EL3 then
        AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoDP);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CGVAP, Clean of Allocation Tags by VA to PoP

The DC CGVAP characteristics are:

Purpose

Clean Allocation Tags in data cache by address to Point of Persistence.

If the memory system does not identify a Point of Persistence, then this instruction behaves as a [DC CGVAC](#).

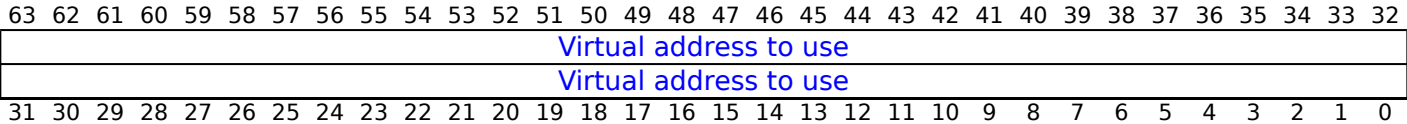
Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC CGVAP are UNDEFINED.

Attributes

DC CGVAP is a 64-bit System instruction.

Field descriptions



Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC CGVAP instruction

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission fault, see 'Permission fault'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CGVAP, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b1100 | 0b011 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCCVAP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoP);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoP);
    elsif PSTATE.EL == EL2 then
        AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoP);
    elsif PSTATE.EL == EL3 then
        AArch64.DC(X[t], CacheType_Tag, CacheOp_Clean, CacheOpScope_PoP);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIGDPAPA, Clean and Invalidate of Data and Allocation Tags by PA to PoPA

The DC CIGDPAPA characteristics are:

Purpose

Clean and Invalidate data and Allocation Tags in data cache by physical address to the Point of Physical Aliasing.

Configuration

This instruction is present only when FEAT_RME is implemented and FEAT_MTE2 is implemented. Otherwise, direct accesses to DC CIGDPAPA are UNDEFINED.

Attributes

DC CIGDPAPA is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------|-----|------|----|----|----|----|----|----|----|----|----|------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| NS | NSE | RES0 | | | | | | | | | | Physical address | | | | | | | | | | | | | | | | | | | |
| Physical address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

NS, bit [63]

Together with the NSE field, this field specifies the target physical address space.

| NSE | NS | Meaning |
|-----|-----|-------------|
| 0b0 | 0b0 | Secure. |
| 0b0 | 0b1 | Non-secure. |
| 0b1 | 0b0 | Root. |
| 0b1 | 0b1 | Realm. |

NSE, bit [62]

Together with the NS field, this field specifies the target physical address space.

For a description of the values derived by evaluating NS and NSE together, see DC CIGDPAPA.NS.

Bits [61:52]

Reserved, RES0.

Bits [51:0]

Physical address to use. No alignment restrictions apply to this PA.

Executing the DC CIGDPAPA instruction

- This instruction is not subject to any translation, permission checks, or granule protection checks.
- This instruction affects all caches in the Outer Shareable shareability domain.

- This instruction has the same ordering, observability, and completion behavior as VA-based cache maintenance instructions issued to the Outer Shareable shareability domain.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CIGDPAPA, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b0111 | 0b1110 | 0b101 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    DC_CIGDPAPA(X[t]);
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIGDSW, Clean and Invalidate of Data and Allocation Tags by Set/Way

The DC CIGDSW characteristics are:

Purpose

Clean and Invalidate data and Allocation Tags in data cache by set/way.

Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC CIGDSW are UNDEFINED.

Attributes

DC CIGDSW is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SetWay | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Level | | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

$A = \text{Log}_2(\text{ASSOCIATIVITY})$, $L = \text{Log}_2(\text{LINELEN})$, $B = (L + S)$, $S = \text{Log}_2(\text{NSETS})$.

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing the DC CIGDSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
 - No cache lines.
 - A single arbitrary cache line.
 - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CIGDSW, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b1110 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_SetWay);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIGDVAC, Clean and Invalidate of Data and Allocation Tags by VA to PoC

The DC CIGDVAC characteristics are:

Purpose

Clean and Invalidate data and Allocation Tags in data cache by address to Point of Coherency.

Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC CIGDVAC are UNDEFINED.

Attributes

DC CIGDVAC is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC CIGDVAC instruction

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission fault.

If FEAT_CMOW is implemented, [HCR_EL2](#).{E2H, TGE} is not {1, 1}, [SCTLR_EL1](#).CMOW is 1, and EL0 access is enabled, when executed at EL0, the instruction has stage 1 read permission to the VA, but does not have stage 1 write permission to the VA, the instruction generates a stage 1 Permission fault.

If FEAT_CMOW is implemented, [HCR_EL2](#).E2H is 1, [SCTLR_EL2](#).CMOW is 1, and EL0 access is enabled, when executed at EL0, the instruction has stage 1 read permission to the VA but does not have stage 1 write permission to the VA, the instruction generates a stage 1 Permission fault.

If FEAT_CMOW is implemented, [HCRX_EL2](#).CMOW is 1, and EL1 or EL0 access is enabled, when executed at EL1 or EL0, the instruction has stage 2 read permission to the VA but does not have stage 2 write permission to the VA, the instruction generates a stage 2 Permission fault.

For more information, see 'Permission fault'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CIGDVAC, <Xt>

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b1110 | 0b101 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCCIVAC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
    elsif PSTATE.EL == EL2 then
        AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
    elsif PSTATE.EL == EL3 then
        AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIGSW, Clean and Invalidate of Allocation Tags by Set/Way

The DC CIGSW characteristics are:

Purpose

Clean and Invalidate Allocation Tags in data cache by set/way.

Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC CIGSW are UNDEFINED.

Attributes

DC CIGSW is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|-------|------|----|----|------|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SetWay | | | | | | | | | | | | | | | | | | | | | | | | Level | | | | RES0 | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | | | | | | | | | | | | | | | | | | | | | Level | | | | RES0 | | | | | | | |

Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

$A = \text{Log}_2(\text{ASSOCIATIVITY})$, $L = \text{Log}_2(\text{LINELEN})$, $B = (L + S)$, $S = \text{Log}_2(\text{NSETS})$.

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing the DC CIGSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
 - No cache lines.
 - A single arbitrary cache line.
 - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CIGSW, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b1110 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_SetWay);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIGVAC, Clean and Invalidate of Allocation Tags by VA to PoC

The DC CIGVAC characteristics are:

Purpose

Clean and Invalidate Allocation Tags in data cache by address to Point of Coherency.

Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC CIGVAC are UNDEFINED.

Attributes

DC CIGVAC is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC CIGVAC instruction

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission fault.

If FEAT_CMOW is implemented, [HCR_EL2](#).{E2H, TGE} is not {1, 1}, [SCTLR_EL1](#).CMOW is 1, and EL0 access is enabled, when executed at EL0, the instruction has stage 1 read permission to the VA, but does not have stage 1 write permission to the VA, the instruction generates a stage 1 Permission fault.

If FEAT_CMOW is implemented, [HCR_EL2](#).E2H is 1, [SCTLR_EL2](#).CMOW is 1, and EL0 access is enabled, when executed at EL0, the instruction has stage 1 read permission to the VA but does not have stage 1 write permission to the VA, the instruction generates a stage 1 Permission fault.

If FEAT_CMOW is implemented, [HCRX_EL2](#).CMOW is 1, and EL1 or EL0 access is enabled, when executed at EL1 or EL0, the instruction has stage 2 read permission to the VA but does not have stage 2 write permission to the VA, the instruction generates a stage 2 Permission fault.

For more information, see 'Permission fault'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CIGVAC, <Xt>

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b1110 | 0b011 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLRL_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCCIVAC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLRL_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
    elsif PSTATE.EL == EL2 then
        AArch64.DC(X[t], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);
    elsif PSTATE.EL == EL3 then
        AArch64.DC(X[t], CacheType_Tag, CacheOp_CleanInvalidate, CacheOpScope_PoC);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIPAPA, Data or unified Cache line Clean and Invalidate by PA to PoPA

The DC CIPAPA characteristics are:

Purpose

Clean and Invalidate data cache by physical address to the Point of Physical Aliasing.

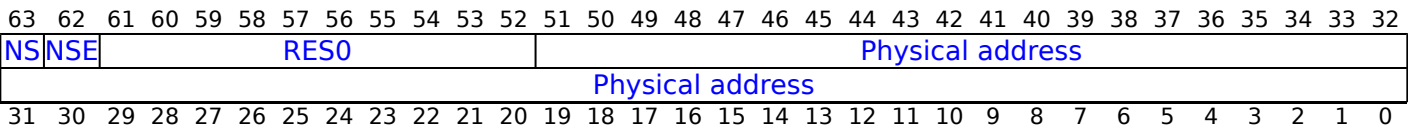
Configuration

This instruction is present only when FEAT_RME is implemented. Otherwise, direct accesses to DC CIPAPA are UNDEFINED.

Attributes

DC CIPAPA is a 64-bit System instruction.

Field descriptions



NS, bit [63]

Together with the NSE field, this field specifies the target physical address space.

| NSE | NS | Meaning |
|-----|-----|-------------|
| 0b0 | 0b0 | Secure. |
| 0b0 | 0b1 | Non-secure. |
| 0b1 | 0b0 | Root. |
| 0b1 | 0b1 | Realm. |

NSE, bit [62]

Together with the NS field, this field specifies the target physical address space.

For a description of the values derived by evaluating NS and NSE together, see DC CIPAPA.NS.

Bits [61:52]

Reserved, RES0.

Bits [51:0]

Physical address to use. No alignment restrictions apply to this PA.

Executing the DC CIPAPA instruction

- This instruction is not subject to any translation, permission checks, or granule protection checks.
- This instruction affects all caches in the Outer Shareable shareability domain.

- This instruction has the same ordering, observability, and completion behavior as VA-based cache maintenance instructions issued to the Outer Shareable shareability domain.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CIPAPA, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b0111 | 0b1110 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    DC_CIPAPA(X[t]);
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CISW, Data or unified Cache line Clean and Invalidate by Set/Way

The DC CISW characteristics are:

Purpose

Clean and Invalidate data cache by set/way.

When FEAT_MTE2 is implemented, this instruction might clean and invalidate Allocation Tags from caches.

Configuration

AArch64 System instruction DC CISW performs the same function as AArch32 System instruction [DCCISW](#).

Attributes

DC CISW is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SetWay | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Level | | RES0 | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

$A = \text{Log}_2(\text{ASSOCIATIVITY})$, $L = \text{Log}_2(\text{LINELEN})$, $B = (L + S)$, $S = \text{Log}_2(\text{NSETS})$.

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing the DC CISW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
 - No cache lines.
 - A single arbitrary cache line.
 - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CISW, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b1110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_SetWay);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CIVAC, Data or unified Cache line Clean and Invalidate by VA to PoC

The DC CIVAC characteristics are:

Purpose

Clean and Invalidate data cache by address to Point of Coherency.

When FEAT_MTE2 is implemented, this instruction might clean and invalidate Allocation Tags from caches.

Configuration

AArch64 System instruction DC CIVAC performs the same function as AArch32 System instruction [DCCIMVAC](#).

Attributes

DC CIVAC is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC CIVAC instruction

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission fault.

When FEAT_CMOW is implemented, [HCR_EL2](#).{E2H, TGE} is not {1, 1}, [SCTLR_EL1](#).CMOW is 1, and EL0 access is implemented, when executed at EL0, the instruction has stage 1 read permission to the VA, but does not have stage 1 write permission to the VA, the instruction generates a stage 1 Permission fault.

When FEAT_CMOW is implemented, [HCR_EL2](#).E2H is 1, [SCTLR_EL2](#).CMOW is 1, and EL0 access is implemented, when executed at EL0, the instruction has stage 1 read permission to the VA but does not have stage 1 write permission to the VA, the instruction generates a stage 1 Permission fault.

When FEAT_CMOW is implemented, [HCRX_EL2](#).CMOW is 1, and EL1 or EL0 access is enabled, when executed at EL1 or EL0, the instruction has stage 2 read permission to the VA but does not have stage 2 write permission to the VA, the instruction generates a stage 2 Permission fault.

For more information, see 'Permission fault'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CIVAC, <Xt>

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b1110 | 0b001 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLRL_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCCIVAC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLRL_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoC);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCIVAC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoC);
    elsif PSTATE.EL == EL2 then
        AArch64.DC(X[t], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoC);
    elsif PSTATE.EL == EL3 then
        AArch64.DC(X[t], CacheType_Data, CacheOp_CleanInvalidate, CacheOpScope_PoC);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CSW, Data or unified Cache line Clean by Set/Way

The DC CSW characteristics are:

Purpose

Clean data cache by set/way.

When FEAT_MTE2 is implemented, this instruction might clean Allocation Tags from caches.

Configuration

AArch64 System instruction DC CSW performs the same function as AArch32 System instruction [DCCSW](#).

Attributes

DC CSW is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SetWay | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Level | | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

$A = \text{Log}_2(\text{ASSOCIATIVITY})$, $L = \text{Log}_2(\text{LINELEN})$, $B = (L + S)$, $S = \text{Log}_2(\text{NSETS})$.

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing the DC CSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
 - No cache lines.
 - A single arbitrary cache line.
 - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CSW, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b1010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_SetWay);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CVAC, Data or unified Cache line Clean by VA to PoC

The DC CVAC characteristics are:

Purpose

Clean data cache by address to Point of Coherency.

When FEAT_MTE2 is implemented, this instruction might clean Allocation Tags from caches.

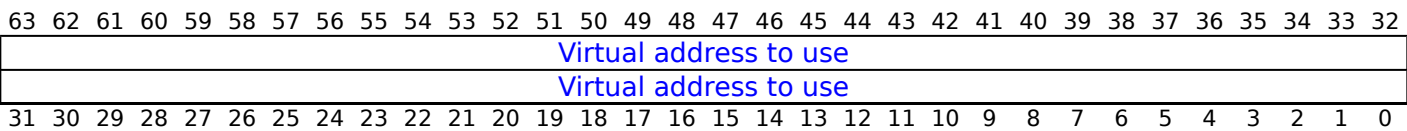
Configuration

AArch64 System instruction DC CVAC performs the same function as AArch32 System instruction [DCCMVAC](#).

Attributes

DC CVAC is a 64-bit System instruction.

Field descriptions



Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC CVAC instruction

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission fault, subject to the constraints described in 'Permission fault'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CVAC, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b1010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCCVAC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoC);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAC == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoC);
    elsif PSTATE.EL == EL2 then
        AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoC);
    elsif PSTATE.EL == EL3 then
        AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoC);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CVADP, Data or unified Cache line Clean by VA to PoDP

The DC CVADP characteristics are:

Purpose

Clean data cache by address to Point of Deep Persistence.

If the memory system does not identify a Point of Deep Persistence, then this instruction behaves as a [DC CVAP](#).

When FEAT_MTE2 is implemented, this instruction might clean Allocation Tags from caches.

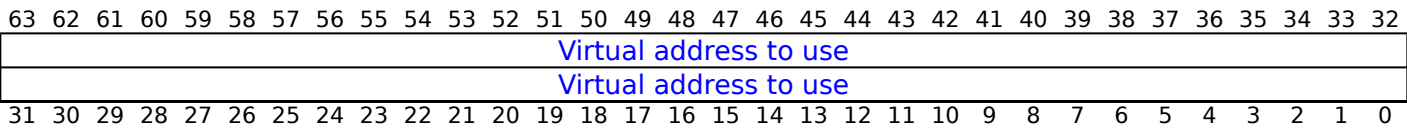
Configuration

This instruction is present only when FEAT_DPB2 is implemented. Otherwise, direct accesses to DC CVADP are UNDEFINED.

Attributes

DC CVADP is a 64-bit System instruction.

Field descriptions



Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC CVADP instruction

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission fault, see 'Permission fault'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CVADP, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b1101 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCCVADP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoDP);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVADP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoDP);
    elsif PSTATE.EL == EL2 then
        AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoDP);
    elsif PSTATE.EL == EL3 then
        AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoDP);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CVAP, Data or unified Cache line Clean by VA to PoP

The DC CVAP characteristics are:

Purpose

Clean data cache by address to Point of Persistence.

If the memory system does not identify a Point of Persistence, then this instruction behaves as a [DC CVAC](#).

When FEAT_MTE2 is implemented, this instruction might clean Allocation Tags from caches.

Configuration

This instruction is present only when FEAT_DPB is implemented. Otherwise, direct accesses to DC CVAP are UNDEFINED.

Attributes

DC CVAP is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC CVAP instruction

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission fault, see 'Permission fault'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CVAP, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b1100 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCCVAP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoP);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoP);
    elsif PSTATE.EL == EL2 then
        AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoP);
    elsif PSTATE.EL == EL3 then
        AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoP);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC CVAU, Data or unified Cache line Clean by VA to PoU

The DC CVAU characteristics are:

Purpose

Clean data cache by address to Point of Unification.

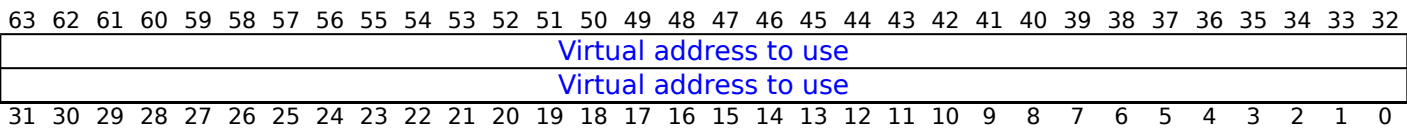
Configuration

AArch64 System instruction DC CVAU performs the same function as AArch32 System instruction [DCCMVAU](#).

Attributes

DC CVAU is a 64-bit System instruction.

Field descriptions



Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC CVAU instruction

If EL0 access is enabled, when executed at EL0, this instruction requires read access permission to the VA, otherwise it generates a Permission fault, subject to the constraints described in 'Permission fault'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC CVAU, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b1011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTL_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPU == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TOCU == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCCVAU == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTL_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoU);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPU == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.TOCU == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCCVAU == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoU);
    elsif PSTATE.EL == EL2 then
        AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoU);
    elsif PSTATE.EL == EL3 then
        AArch64.DC(X[t], CacheType_Data, CacheOp_Clean, CacheOpScope_PoU);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC GVA, Data Cache set Allocation Tag by VA

The DC GVA characteristics are:

Purpose

Write a value to the Allocation Tags of a naturally aligned block of N bytes, where the size of N is identified in [DCZID_EL0](#). The Allocation Tag used is determined by the input address.

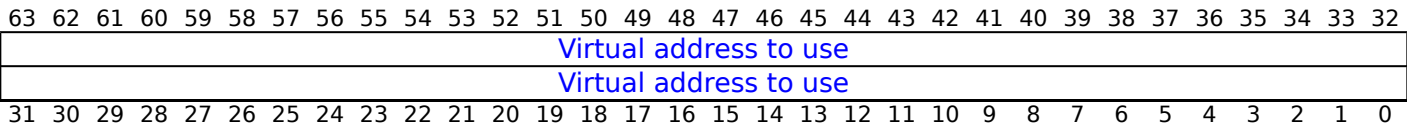
Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC GVA are UNDEFINED.

Attributes

DC GVA is a 64-bit System instruction.

Field descriptions



Bits [63:0]

Virtual address to use. There is no alignment restriction on the address within the block of N bytes that is used.

Executing the DC GVA instruction

When this instruction is executed, it can generate memory faults or watchpoints which are prioritized in the same way as other memory-related faults or watchpoints. If a synchronous data abort fault or a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is not set.

If the memory region being modified is any type of Device memory, this instruction can give an alignment fault that is prioritized in the same way as other alignment faults that are determined by the memory type.

This instruction applies to Normal memory regardless of cacheability attributes.

This instruction behaves as a set of stores to each Allocation Tag within the block being accessed, and so it:

- Generates a Permission fault if the translation system does not permit writes to the locations.
- Requires the same considerations for ordering and the management of coherency as any other store instructions.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC GVA, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b0100 | 0b011 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.DZE == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TDZ == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCZVA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.DZE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.MemZero(X[t], CacheType_Tag);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TDZ == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCZVA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.MemZero(X[t], CacheType_Tag);
    elsif PSTATE.EL == EL2 then
        AArch64.MemZero(X[t], CacheType_Tag);
    elsif PSTATE.EL == EL3 then
        AArch64.MemZero(X[t], CacheType_Tag);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC GZVA, Data Cache set Allocation Tags and Zero by VA

The DC GZVA characteristics are:

Purpose

Zero data and write a value to the Allocation Tags of a naturally aligned block of N bytes, where the size of N is identified in [DCZID_ELO](#). The Allocation Tag used is determined by the input address.

Configuration

This instruction is present only when FEAT_MTE is implemented. Otherwise, direct accesses to DC GZVA are UNDEFINED.

Attributes

DC GZVA is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual address to use. There is no alignment restriction on the address within the block of N bytes that is used.

Executing the DC GZVA instruction

When this instruction is executed, it can generate memory faults or watchpoints which are prioritized in the same way as other memory-related faults or watchpoints. If a synchronous data abort fault or a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is not set.

If the memory region being zeroed is any type of Device memory, this instruction can give an alignment fault which is prioritized in the same way as other alignment faults that are determined by the memory type.

This instruction applies to Normal memory regardless of cacheability attributes.

This instruction behaves as a set of Stores to each byte and Allocation tag within the block being accessed, and so it:

- Generates a Permission fault if the translation system does not permit writes to the locations.
- Requires the same considerations for ordering and the management of coherency as any other store instructions.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC GZVA, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b0100 | 0b100 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.DZE == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TDZ == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCZVA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.DZE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.MemZero(X[t], CacheType_Data_Tag);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TDZ == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCZVA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.MemZero(X[t], CacheType_Data_Tag);
    elsif PSTATE.EL == EL2 then
        AArch64.MemZero(X[t], CacheType_Data_Tag);
    elsif PSTATE.EL == EL3 then
        AArch64.MemZero(X[t], CacheType_Data_Tag);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC IGDSW, Invalidate of Data and Allocation Tags by Set/Way

The DC IGDSW characteristics are:

Purpose

Invalidate data and Allocation Tags in data cache by set/way.

Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC IGDSW are UNDEFINED.

Attributes

DC IGDSW is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|--|--|------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SetWay | | | | | | | | | | | | | | | | | | | | | | | | Level | | | | | | | | | | RES0 | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

$A = \text{Log}_2(\text{ASSOCIATIVITY})$, $L = \text{Log}_2(\text{LINELEN})$, $B = (L + S)$, $S = \text{Log}_2(\text{NSETS})$.

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing the DC IGDSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
 - No cache lines.
 - A single arbitrary cache line.
 - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC IGDSW, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b0110 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Invalidate, CacheOpScope_SetWay);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC IGDVAC, Invalidate of Data and Allocation Tags by VA to PoC

The DC IGDVAC characteristics are:

Purpose

Invalidate data and Allocation Tags in data cache by address to Point of Coherency.

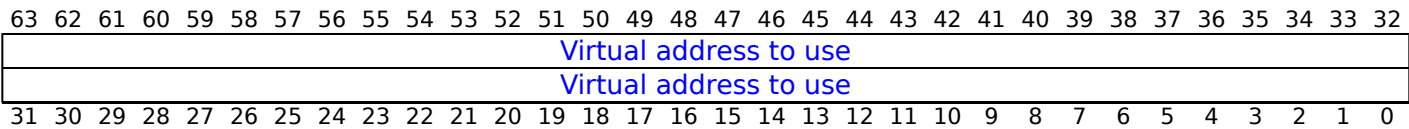
Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC IGDVAC are UNDEFINED.

Attributes

DC IGDVAC is a 64-bit System instruction.

Field descriptions



Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC IGDVAC instruction

When the instruction is executed, it can generate a watchpoint, which is prioritized in the same way as other watchpoints. If a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is set to 1.

This instruction requires write access permission to the VA, otherwise it generates a Permission fault, subject to the constraints described in 'Permission fault'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC IGDVAC, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b0110 | 0b101 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t], CacheType_Data_Tag, CacheOp_Invalidate, CacheOpScope_PoC);
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC IGSW, Invalidate of Allocation Tags by Set/Way

The DC IGSW characteristics are:

Purpose

Invalidate Allocation Tags in data cache by set/way.

Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC IGSW are UNDEFINED.

Attributes

DC IGSW is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | |
| SetWay | | | | | | | | | | | | | | | | | | | | | | | | | | | Level | | | | RES0 | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

$A = \text{Log}_2(\text{ASSOCIATIVITY})$, $L = \text{Log}_2(\text{LINELEN})$, $B = (L + S)$, $S = \text{Log}_2(\text{NSETS})$.

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing the DC IGSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
 - No cache lines.
 - A single arbitrary cache line.
 - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC IGSW, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b0110 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t], CacheType_Tag, CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t], CacheType_Tag, CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t], CacheType_Tag, CacheOp_Invalidate, CacheOpScope_SetWay);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC IGVAC, Invalidate of Allocation Tags by VA to PoC

The DC IGVAC characteristics are:

Purpose

Invalidate Allocation Tags in data cache by address to Point of Coherency.

Configuration

This instruction is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to DC IGVAC are UNDEFINED.

Attributes

DC IGVAC is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC IGVAC instruction

When the instruction is executed, it can generate a watchpoint, which is prioritized in the same way as other watchpoints. If a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is set to 1.

This instruction requires write access permission to the VA, otherwise it generates a Permission fault, subject to the constraints described in 'Permission fault'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC IGVAC, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b0110 | 0b011 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t], CacheType_Tag, CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t], CacheType_Tag, CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t], CacheType_Tag, CacheOp_Invalidate, CacheOpScope_PoC);
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC ISW, Data or unified Cache line Invalidate by Set/Way

The DC ISW characteristics are:

Purpose

Invalidate data cache by set/way.

When FEAT_MTE2 is implemented, this instruction might invalidate Allocation Tags from caches. When it invalidates Allocation Tags from caches, it also cleans them.

Configuration

AArch64 System instruction DC ISW performs the same function as AArch32 System instruction [DCISW](#).

Attributes

DC ISW is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SetWay | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Level | | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

$A = \text{Log}_2(\text{ASSOCIATIVITY})$, $L = \text{Log}_2(\text{LINELEN})$, $B = (L + S)$, $S = \text{Log}_2(\text{NSETS})$.

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing the DC ISW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED.
- The instruction performs cache maintenance on one of:
 - No cache lines.
 - A single arbitrary cache line.
 - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC ISW, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b0110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TSW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCISW == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t], CacheType_Data, CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t], CacheType_Data, CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t], CacheType_Data, CacheOp_Invalidate, CacheOpScope_SetWay);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC IVAC, Data or unified Cache line Invalidate by VA to PoC

The DC IVAC characteristics are:

Purpose

Invalidate data cache by address to Point of Coherency.

When FEAT_MTE2 is implemented, this instruction might invalidate Allocation Tags from caches. When it invalidates Allocation Tags from caches, it also cleans them.

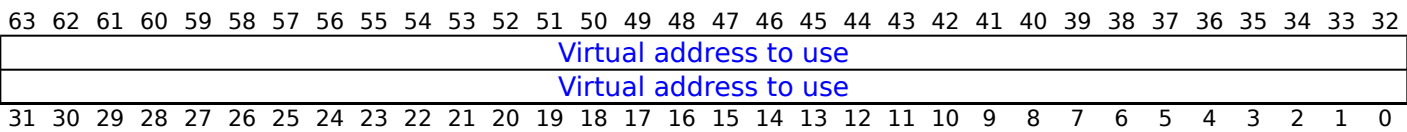
Configuration

AArch64 System instruction DC IVAC performs the same function as AArch32 System instruction [DCIMVAC](#).

Attributes

DC IVAC is a 64-bit System instruction.

Field descriptions



Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DC IVAC instruction

When the instruction is executed, it can generate a watchpoint, which is prioritized in the same way as other watchpoints. If a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is set to 1.

This instruction requires write access permission to the VA, otherwise it generates a Permission fault, subject to the constraints described in 'Permission fault'.

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The data cache maintenance instruction (DC)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC IVAC, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b0110 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPCP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCIVAC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.DC(X[t], CacheType_Data, CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch64.DC(X[t], CacheType_Data, CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch64.DC(X[t], CacheType_Data, CacheOp_Invalidate, CacheOpScope_PoC);
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DC ZVA, Data Cache Zero by VA

The DC ZVA characteristics are:

Purpose

Zero data cache by address. Zeroes a naturally aligned block of N bytes, where the size of N is identified in [DCZID_EL0](#).

Configuration

There are no configuration notes.

Attributes

DC ZVA is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual address to use. There is no alignment restriction on the address within the block of N bytes that is used.

Executing the DC ZVA instruction

When this instruction is executed, it can generate memory faults or watchpoints which are prioritized in the same way as other memory-related faults or watchpoints. If a synchronous data abort fault or a watchpoint is generated, the CM bit in the ESR_ELx.ISS field is set to 0.

If the memory region being zeroed is any type of Device memory, this instruction can give an Alignment fault which is prioritized in the same way as other Alignment faults that are determined by the memory type.

This instruction applies to Normal memory regardless of cacheability attributes.

This instruction behaves as a set of Stores to each byte within the block being accessed, and so it:

- Generates a Permission fault if the translation system does not permit writes to the locations.
- Requires the same considerations for ordering and the management of coherency as any other store instructions.

Accesses to this instruction use the following encodings in the System instruction encoding space:

DC ZVA, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.DZE == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TDZ == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DCZVA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.DZE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.MemZero(X[t], CacheType_Data);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TDZ == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DCZVA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.MemZero(X[t], CacheType_Data);
    elsif PSTATE.EL == EL2 then
        AArch64.MemZero(X[t], CacheType_Data);
    elsif PSTATE.EL == EL3 then
        AArch64.MemZero(X[t], CacheType_Data);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCZID_EL0, Data Cache Zero ID register

The DCZID_EL0 characteristics are:

Purpose

Indicates the block size that is written with byte values of 0 by the [DC ZVA](#) (Data Cache Zero by Address) System instruction.

If FEAT_MTE is implemented, this register also indicates the granularity at which the [DC GVA](#) and [DC GZVA](#) instructions write.

Configuration

There are no configuration notes.

Attributes

DCZID_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:5]

Reserved, RES0.

DZP, bit [4]

Data Zero Prohibited. This field indicates whether use of [DC ZVA](#) instructions is permitted or prohibited.

If FEAT_MTE is implemented, this field also indicates whether use of the [DC GVA](#) and [DC GZVA](#) instructions are permitted or prohibited.

| DZP | Meaning |
|-----|------------------------------|
| 0b0 | Instructions are permitted. |
| 0b1 | Instructions are prohibited. |

The value read from this field is governed by the access state and the values of the [HCR_EL2.TDZ](#) and [SCTLR_EL1.DZE](#) bits.

BS, bits [3:0]

Log₂ of the block size in words. The maximum size supported is 2KB (value == 9).

Accessing DCZID_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DCZID_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0000 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGTR_EL2.DCZID_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return DCZID_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.DCZID_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return DCZID_EL0;
elsif PSTATE.EL == EL2 then
    return DCZID_EL0;
elsif PSTATE.EL == EL3 then
    return DCZID_EL0;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DISR_EL1, Deferred Interrupt Status Register

The DISR_EL1 characteristics are:

Purpose

Records that an SError interrupt has been consumed by an ESB instruction.

Configuration

AArch64 System register DISR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DISR\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to DISR_EL1 are UNDEFINED.

Attributes

DISR_EL1 is a 64-bit register.

Field descriptions

When DISR_EL1.IDS == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|----|----|----|-----|------|----|----|----|----|----|----|----|-----|----|----|------|----|----|------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| A | RES0 | | | | IDS | RES0 | | | | | | | | AET | | EA | RES0 | | | DFSC | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers an asynchronous SError interrupt. If the implementation does not include any sources of SError interrupt that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:25]

Reserved, RES0.

IDS, bit [24]

Indicates the deferred SError interrupt type.

| IDS | Meaning |
|-----|---|
| 0b0 | Deferred error uses architecturally-defined format. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:13]

Reserved, RES0.

AET, bits [12:10]

Asynchronous Error Type. See the description of ESR_ELx.AET for an SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort Type. See the description of ESR_ELx.EA for an SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

DFSC, bits [5:0]

Fault Status Code. See the description of ESR_ELx.DFSC for an SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When DISR_EL1.IDS == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|----|----|-----|-----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | |
| A | RES0 | | | | | | IDS | ISS | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers an asynchronous SError interrupt. If the implementation does not include any sources of SError interrupt that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:25]

Reserved, RES0.

IDS, bit [24]

Indicates the deferred SError interrupt type.

| IDS | Meaning |
|-----|--|
| 0b1 | Deferred error uses IMPLEMENTATION DEFINED format. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]

IMPLEMENTATION DEFINED syndrome. See the description of ESR_ELx[23:0] for an SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DISR_EL1

An indirect write to DISR_EL1 made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR_EL1 occurring in program order after the ESB instruction.

DISR_EL1 is RAZ/WI if EL3 is implemented, the PE is in Non-debug state, [SCR_EL3.EA](#) == 1, and any of the following apply:

- At EL2.
- At EL1 and (([SCR_EL3.NS](#) == 0 && [SCR_EL3.EEL2](#) == 0) || [HCR_EL2.AMO](#) == 0).

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DISR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AMO == '1' then
        return VDISR_EL2;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return Zeros();
    else
        return DISR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return Zeros();
    else
        return DISR_EL1;
elsif PSTATE.EL == EL3 then
    return DISR_EL1;

```

MSR DISR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AMO == '1' then
        VDISR_EL2 = X[t];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        //no operation
    else
        DISR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        //no operation
    else
        DISR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    DISR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DIT, Data Independent Timing

The DIT characteristics are:

Purpose

Allows access to the Data Independent Timing bit.

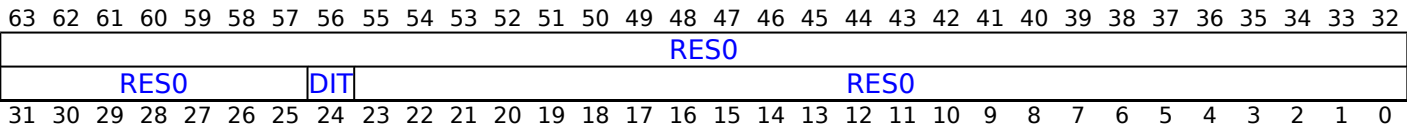
Configuration

This register is present only when FEAT_DIT is implemented. Otherwise, direct accesses to DIT are UNDEFINED.

Attributes

DIT is a 64-bit register.

Field descriptions



Bits [63:25]

Reserved, RES0.

DIT, bit [24]

Data Independent Timing.

| DIT | Meaning |
|-----|---|
| 0b0 | The architecture makes no statement about the timing properties of any instructions. |
| 0b1 | The architecture requires that: <ul style="list-style-type: none">The timing of every load and store instruction is insensitive to the value of the data being loaded or stored.For certain data processing instructions, the instruction takes a time which is independent of:<ul style="list-style-type: none">The values of the data supplied in any of its registers.The values of the NZCV flags.For certain data processing instructions, the response of the instruction to asynchronous exceptions does not vary based on:<ul style="list-style-type: none">The values of the data supplied in any of its registers.The values of the NZCV flags. |

The data processing instructions affected by this bit are:

- All cryptographic instructions. These instructions are:
 - AESD, AESE, AESIMC, AESMC, SHA1C, SHA1H, SHA1M, SHA1P, SHA1SU0, SHA1SU1, SHA256H, SHA256H2, SHA256SU0, SHA256SU1, SHA512H, SHA512H2, SHA512SU0, SHA512SU1, EOR3, RAX1, XAR, BCAX, SM3SS1, SM3TT1A, SM3TT1B, SM3TT2A, SM3TT2B, SM3PARTW1, SM3PARTW2, SM4E, and SM4EKEY.
- A subset of those instructions which use the general-purpose register file. These instructions are:

- ADC, ADCS, ADD, ADDS, AND, ANDS, ASR, ASRV, BFC, BFI, BFM, BFXIL, BIC, BICS, CCMN, CCMP, CFINV, CINC, CINV, CLS, CLZ, CMN, CMP, CNEG, CSEL, CSET, CSETM, CSINC, CSINV, CSNEG, EON, EOR, EXTR, LSL, LSLV, LSR, LSRV, MADD, MNEG, MOV, MOVK, MOVN, MOVZ, MSUB, MUL, MVN, NEG, NEGS, NGC, NGCS, NOP, ORN, ORR, RBIT, RET, REV, REV16, REV32, REV64, RMIF, ROR, RORV, SBC, SBCS, SBFIZ, SBFM, SBFX, SETF8, SETF16, SMADDL, SMNEGL, SMSUBL, SMULH, SMULL, SUB, SUBS, SXTB, SXTH, SXTW, TST, UBFIZ, UBFM, UBFX, UMADDL, UMNEGL, UMSUBL, UMULH, UMULL, UXTB, and UXTH.
- If FEAT_CRC32 is implemented, CRC32B, CRC32H, CRC32W, CRC32X, CRC32CB, CRC32CH, CRC32CW, and CRC32CX.
- A subset of those instructions which use the SIMD&FP register file. These instructions are:
 - ABS, ADD, ADDHN, ADDHN2, ADDP, ADDV, AND, BIC, BIF, BIT, BSL, CLS, CLZ, CMEQ, CMGE, CMGT, CMHI, CMHS, CMLE, CMLT, CMTST, CNT, DUP, EOR, EXT, FCSEL, INS, MLA, MLS, MOV, MOVI, MUL, MVN, MVNI, NEG, NOT, ORN, ORR, PMUL, PMULL, PMULL2, RADDHN, RADDHN2, RBIT, REV16, REV32, RSHRN, RSHRN2, RSUBHN, RSUBHN2, SABA, SABD, SABAL, SABAL2, SABDL, SABDL2, SADALP, SADDL, SADDL2, SADDLP, SADDLV, SADDW, SADDW2, SHADD, SHL, SHLL, SHLL2, SHRN, SHRN2, SHSUB, SLI, SMAX, SMAXP, SMAXV, SMIN, SMINP, SMINV, SMLAL, SMLAL2, SMLSL, SMLSL2, SMOV, SMULL, SMULL2, SRI, SSHL, SSHLL, SSHLL2, SSHR, SSRA, SSUBL, SSUBL2, SSUBW, SSUBW2, SUB, SUBHN, SUBHN2, SXTL, SXTL2, TBL, TBX, TRN1, TRN2, UABA, UABAL, UABAL2, UABD, UABDL, UABDL2, UADALP, UADDL, UADDL2, UADDLP, UADDLV, UADDW, UADDW2, UHADD, UHSUB, UMAX, UMAXP, UMAXV, UMIN, UMINP, UMINV, UMLAL, UMLAL2, UMLSL, UMOV, UMLSL2, UMULL, UMULL2, USHL, USHLL, USHLL2, USHR, USRA, USUBL, USUBL2, USUBW, USUBW2, UXTL, UXTL2, UZP1, UZP2, XTN, XTN2, ZIP1, and ZIP2.

Note

The architecture makes no statement about the timing properties when the PSTATE.DIT bit is not set. However, it is likely that many of these instructions have timing that is invariant of the data in many situations.

In particular, Arm strongly recommends that the Armv8.3 pointer authentication instructions do not have their timing dependent on the key value used in the pointer authentication in all cases, regardless of the PSTATE.DIT bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [23:0]

Reserved, RES0.

Accessing DIT

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DIT

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b101 |

```

if PSTATE.EL == EL0 then
    return Zeros(39):PSTATE.DIT:Zeros(24);
elsif PSTATE.EL == EL1 then
    return Zeros(39):PSTATE.DIT:Zeros(24);
elsif PSTATE.EL == EL2 then
    return Zeros(39):PSTATE.DIT:Zeros(24);
elsif PSTATE.EL == EL3 then
    return Zeros(39):PSTATE.DIT:Zeros(24);

```

MSR DIT, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b101 |

```

if PSTATE.EL == EL0 then
    PSTATE.DIT = X[t]<24>;
elsif PSTATE.EL == EL1 then
    PSTATE.DIT = X[t]<24>;
elsif PSTATE.EL == EL2 then
    PSTATE.DIT = X[t]<24>;
elsif PSTATE.EL == EL3 then
    PSTATE.DIT = X[t]<24>;

```

MSR DIT, #<imm>

| op0 | op1 | CRn | op2 |
|------|-------|--------|-------|
| 0b00 | 0b011 | 0b0100 | 0b010 |

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DLR_EL0, Debug Link Register

The DLR_EL0 characteristics are:

Purpose

In Debug state, holds the address to restart from.

Configuration

AArch64 System register DLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [DLR\[31:0\]](#).

Attributes

DLR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Restart address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Restart address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Restart address.

Accessing DLR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DLR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0101 | 0b001 |

```
if !Halted() then
    UNDEFINED;
else
    return DLR_EL0;
```

MSR DLR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0101 | 0b001 |

```
if !Halted() then
    UNDEFINED;
else
    DLR_EL0 = X[t];
```


DSPSR_EL0, Debug Saved Program Status Register

The DSPSR_EL0 characteristics are:

Purpose

Holds the saved process state for Debug state. On entering Debug state, PSTATE information is written to this register. On exiting Debug state, values are copied from this register to PSTATE.

Configuration

AArch64 System register DSPSR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [DSPSR\[31:0\]](#).

Attributes

DSPSR_EL0 is a 64-bit register.

Field descriptions

When AArch32 is supported and exiting Debug state to AArch32 state:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|---------|-----|------|-----|----|----|----|---------|----|----|----|----|----|------|--------|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | Z | C | V | Q | IT[1:0] | DIT | SSBS | PAN | SS | IL | GE | IT[7:2] | E | A | I | F | T | M[4] | M[3:0] | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Copied to PSTATE.N on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Copied to PSTATE.Z on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Copied to PSTATE.C on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Copied to PSTATE.V on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Copied to PSTATE.Q on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Copied to PSTATE.IT on exiting Debug state.

DPSR_EL0.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is DPSR_EL0[26:25].
- IT[7:2] is DPSR_EL0[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Copied to PSTATE.DIT on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Copied to PSTATE.SSBS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Copied to PSTATE.PAN on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Copied to PSTATE.SS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Copied to PSTATE.IL on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Copied to PSTATE.GE on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Copied to PSTATE.E on exiting Debug state.

If the implementation does not support big-endian operation, DSPSR_EL0.E is RES0. If the implementation does not support little-endian operation, DSPSR_EL0.E is RES1. On exiting Debug state, if the implementation does not support big-endian operation at the Exception level being returned to, DSPSR_EL0.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, DSPSR_EL0.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Copied to PSTATE.A on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Copied to PSTATE.I on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Copied to PSTATE.F on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Copied to PSTATE.T on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4], bit [4]

Execution state. Copied to PSTATE.nRW on exiting Debug state.

| M[4] | Meaning |
|------|--------------------------|
| 0b1 | AArch32 execution state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch32 Mode. Copied to PSTATE.M[3:0] on exiting Debug state.

| M[3:0] | Meaning |
|--------|-------------|
| 0b0000 | User. |
| 0b0001 | FIQ. |
| 0b0010 | IRQ. |
| 0b0011 | Supervisor. |
| 0b0110 | Monitor. |
| 0b0111 | Abort. |
| 0b1010 | Hyp. |
| 0b1011 | Undefined. |
| 0b1111 | System. |

Other values are reserved. If DPSR_EL0.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, exiting Debug state is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When AArch64 is supported and entering or exiting Debug state from or to AArch64 state:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|------|--------|--------|----|----|------|----|----|----|----|----|----|----|----|----|--------|------|-------|----|----|----|----|------|------|--------|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | Z | C | V | RES0 | TCODIT | UAOPAN | SS | IL | RES0 | | | | | | | | | | ALLINT | SSBS | BTYPE | D | A | I | F | RES0 | M[4] | M[3:0] | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on entering Debug state, and copied to PSTATE.N on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on entering Debug state, and copied to PSTATE.Z on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on entering Debug state, and copied to PSTATE.C on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on entering Debug state, and copied to PSTATE.V on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

TCO, bit [25]

When FEAT_MTE is implemented:

Tag Check Override. Set to the value of PSTATE.TCO on entering Debug state, and copied to PSTATE.TCO on exiting Debug state.

When FEAT_MTE2 is not implemented, it is CONSTRAINED UNPREDICTABLE whether this field is RES0 or behaves as if FEAT_MTE is implemented.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on entering Debug state, and copied to PSTATE.DIT on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UAO, bit [23]

When FEAT_UAO is implemented:

User Access Override. Set to the value of PSTATE.UAO on entering Debug state, and copied to PSTATE.UAO on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on entering Debug state, and copied to PSTATE.PAN on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on entering Debug state, and conditionally copied to PSTATE.SS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on entering Debug state, and copied to PSTATE.IL on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:14]

Reserved, RES0.

ALLINT, bit [13]**When FEAT_NMI is implemented:**

All IRQ or FIQ interrupts mask. Set to the value of PSTATE.ALLINT on entering Debug state, and copied to PSTATE.ALLINT on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [12]**When FEAT_SSBS is implemented:**

Speculative Store Bypass. Set to the value of PSTATE.SSBS on entering Debug state, and copied to PSTATE.SSBS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BTYPE, bits [11:10]**When FEAT_BTI is implemented:**

Branch Type Indicator. Set to the value of PSTATE.BTYPE on entering Debug state, and copied to PSTATE.BTYPE on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D, bit [9]

Debug exception mask. Set to the value of PSTATE.D on entering Debug state, and copied to PSTATE.D on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

Error interrupt mask. Set to the value of PSTATE.A on entering Debug state, and copied to PSTATE.A on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on entering Debug state, and copied to PSTATE.I on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on entering Debug state, and copied to PSTATE.F on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

M[4], bit [4]

Execution state. Set to 0b0, the value of PSTATE.nRW, on entering Debug state from AArch64 state, and copied to PSTATE.nRW on exiting Debug state.

| M[4] | Meaning |
|------|--------------------------|
| 0b0 | AArch64 execution state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch64 Exception level and selected Stack Pointer.

| M[3:0] | Meaning |
|--------|---------|
| 0b0000 | EL0t. |
| 0b0100 | EL1t. |
| 0b0101 | EL1h. |
| 0b1000 | EL2t. |
| 0b1001 | EL2h. |
| 0b1100 | EL3t. |
| 0b1101 | EL3h. |

Other values are reserved. If DPSR_EL0.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, exiting Debug state is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The bits in this field are interpreted as follows:

- M[3:2] is set to the value of PSTATE.EL on entering Debug state and copied to PSTATE.EL on exiting Debug state.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on entering Debug state and copied to PSTATE.SP on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DPSR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, DSPSR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0101 | 0b000 |

```

if !Halted() then
    UNDEFINED;
else
    return DSPSR_EL0;

```

MSR DSPSR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0101 | 0b000 |

```

if !Halted() then
    UNDEFINED;
else
    DSPSR_EL0 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DVP RCTX, Data Value Prediction Restriction by Context

The DVP RCTX characteristics are:

Purpose

Data Value Prediction Restriction by Context applies to all Data Value Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

Data value predictions determined by the actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control speculative execution occurring after the instruction is complete and synchronized.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

Configuration

This instruction is present only when FEAT_SPECRES is implemented. Otherwise, direct accesses to DVP RCTX are UNDEFINED.

Attributes

DVP RCTX is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|------|----|----|----|----|----|----|----|----|-------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | GVMID | VMID | | | | | | | | | | | | | | | | |
| RES0 | | | | NSENS | EL | RES0 | | | | | | | | | GASID | ASID | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:49]

Reserved, RES0.

GVMID, bit [48]

Execution of this instruction applies to all VMIDs or a specified VMID.

| GVMID | Meaning |
|-------|---|
| 0b0 | Applies to specified VMID for an EL0 or EL1 target execution context. |
| 0b1 | Applies to all VMIDs for an EL0 or EL1 target execution context. |

For target execution contexts other than EL0 or EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

VMID, bits [47:32]

Only applies when bit[48] is 0 and the target execution context is either:

- EL1.
- EL0 when ([HCR_EL2.E2H==0](#) or [HCR_EL2.TGE==0](#)).

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0 and ([HCR_EL2.E2H==0](#) or [HCR_EL2.TGE==0](#)), this field is treated as the current VMID.

When the instruction is executed at EL0 and ([HCR_EL2.E2H==1](#) and [HCR_EL2.TGE==1](#)), this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

If the implementation supports 16 bits of VMID, then the upper 8 bits of the VMID must be written to 0 by software when the context being affected only uses 8 bits.

Bits [31:28]

Reserved, RES0.

NSE, bit [27]

When FEAT_RME is implemented:

Together with the NS field, selects the Security state.

For a description of the values derived by evaluating NS and NSE together, see DVP_RCTX.NS.

Otherwise:

Reserved, RES0.

NS, bit [26]

When FEAT_RME is implemented:

Together with the NSE field, selects the Security state. Defined values are:

| NSE | NS | Meaning |
|-----|-----|-------------|
| 0b0 | 0b0 | Secure. |
| 0b0 | 0b1 | Non-secure. |
| 0b1 | 0b0 | Root. |
| 0b1 | 0b1 | Realm. |

Some Effective values are determined by the current Security state:

- When executed in Secure state, the Effective value of NSE is 0.
- When executed in Non-secure state, the Effective value of {NSE, NS} is {0, 1}.
- When executed in Realm state, the Effective value of {NSE, NS} is {1, 1}.

An instruction with an EL field that has a value other than 0b11 (EL3) is treated as a NOP when executed at EL3 with DVP_RCTX.{NSE, NS} == {1, 0}.

Otherwise:

Security State. Defined values are:

| NS | Meaning |
|-----|-------------------|
| 0b0 | Secure state. |
| 0b1 | Non-secure state. |

When executed in Non-secure state, the Effective value of NS is 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

| EL | Meaning |
|------|---------|
| 0b00 | EL0. |
| 0b01 | EL1. |
| 0b10 | EL2. |
| 0b11 | EL3. |

If the instruction is executed at an Exception level lower than the specified level, this instruction is treated as a NOP.

Bits [23:17]

Reserved, RES0.

GASID, bit [16]

Execution of this instruction applies to all ASIDs or a specified ASID.

| GASID | Meaning |
|-------|--|
| 0b0 | Applies to specified ASID for an EL0 target execution context. |
| 0b1 | Applies to all ASID for an EL0 target execution context. |

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field has an Effective value of 0.

ASID, bits [15:0]

Only applies for an EL0 target execution context and when bit[16] is 0.

Otherwise this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being affected only uses 8 bits.

Executing the DVP RCTX instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

DVP RCTX, <Xt>

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b0011 | 0b101 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.DVPRCTX == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.EnRCTX == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.RestrictPrediction(X[t], RestrictType_DataValue);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.NV == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.DVPRCTX == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.RestrictPrediction(X[t], RestrictType_DataValue);
    elsif PSTATE.EL == EL2 then
        AArch64.RestrictPrediction(X[t], RestrictType_DataValue);
    elsif PSTATE.EL == EL3 then
        AArch64.RestrictPrediction(X[t], RestrictType_DataValue);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ELR_EL1, Exception Link Register (EL1)

The ELR_EL1 characteristics are:

Purpose

When taking an exception to EL1, holds the address to return to.

Configuration

There are no configuration notes.

Attributes

ELR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Return address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Return address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Return address.

An exception return from EL1 using AArch64 makes ELR_EL1 become UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ELR_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic ELR_EL1 or ELR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ELR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x230];
    else
        return ELR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return ELR_EL2;
    else
        return ELR_EL1;
elsif PSTATE.EL == EL3 then
    return ELR_EL1;

```

MSR ELR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x230] = X[t];
    else
        ELR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        ELR_EL2 = X[t];
    else
        ELR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ELR_EL1 = X[t];

```

MRS <Xt>, ELR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0100 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x230];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return ELR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return ELR_EL1;
    else
        UNDEFINED;

```

MSR ELR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0100 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x230] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        ELR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        ELR_EL1 = X[t];
    else
        UNDEFINED;

```

MRS <Xt>, ELR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0100 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return ELR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return ELR_EL2;
elsif PSTATE.EL == EL3 then
    return ELR_EL2;

```

MSR ELR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0100 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        ELR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ELR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    ELR_EL2 = X[t];

```


ELR_EL2, Exception Link Register (EL2)

The ELR_EL2 characteristics are:

Purpose

When taking an exception to EL2, holds the address to return to.

Configuration

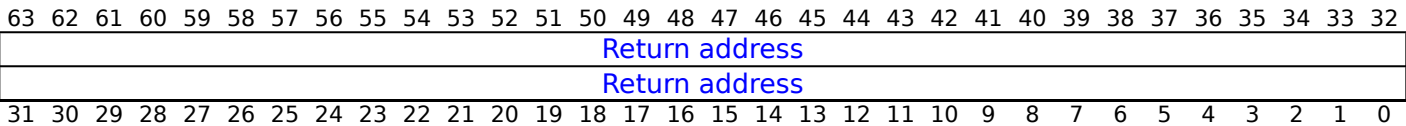
AArch64 System register ELR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [ELR_hyp\[31:0\]](#).

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ELR_EL2 is a 64-bit register.

Field descriptions



Bits [63:0]

Return address.

An exception return from EL2 using AArch64 makes ELR_EL2 become UNKNOWN.

When EL2 is in AArch32 Execution state and an exception is taken from EL0, EL1, or EL2 to EL3 and AArch64 execution, the upper 32-bits of ELR_EL2 are either set to 0 or hold the same value that they did before AArch32 execution. Which option is adopted is determined by an implementation, and might vary dynamically within an implementation. Correspondingly software must regard the value as being an UNKNOWN choice between the two values.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ELR_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic ELR_EL2 or ELR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ELR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0100 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return ELR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return ELR_EL2;
elsif PSTATE.EL == EL3 then
    return ELR_EL2;

```

MSR ELR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0100 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        ELR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ELR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    ELR_EL2 = X[t];

```

MRS <Xt>, ELR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x230];
    else
        return ELR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return ELR_EL2;
    else
        return ELR_EL1;
elsif PSTATE.EL == EL3 then
    return ELR_EL1;

```

MSR ELR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0000 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x230] = X[t];
    else
        ELR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        ELR_EL2 = X[t];
    else
        ELR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ELR_EL1 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ELR_EL3, Exception Link Register (EL3)

The ELR_EL3 characteristics are:

Purpose

When taking an exception to EL3, holds the address to return to.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to ELR_EL3 are UNDEFINED.

Attributes

ELR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Return address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Return address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Return address.

An exception return from EL3 using AArch64 makes ELR_EL3 become UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ELR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ELR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0100 | 0b0000 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return ELR_EL3;
```

MSR ELR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|

ELR_EL3, Exception Link Register (EL3)

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0100 | 0b0000 | 0b001 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    ELR_EL3 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRIDR_EL1, Error Record ID Register

The ERRIDR_EL1 characteristics are:

Purpose

Defines the highest numbered index of the error records that can be accessed through the Error Record System registers.

Configuration

AArch64 System register ERRIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ERRIDR\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERRIDR_EL1 are UNDEFINED.

Attributes

ERRIDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | NUM | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

NUM, bits [15:0]

Highest numbered index of the records that can be accessed through the Error Record System registers plus one. Zero indicates no records can be accessed through the Error Record System registers.

Each implemented record is owned by a node. A node might own multiple records.

Accessing ERRIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERRIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ERRIDR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ERRIDR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ERRIDR_EL1;
elsif PSTATE.EL == EL3 then
    return ERRIDR_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRSELR_EL1, Error Record Select Register

The ERRSELR_EL1 characteristics are:

Purpose

Selects an error record to be accessed through the Error Record System registers.

Configuration

AArch64 System register ERRSELR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ERRSELR\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERRSELR_EL1 are UNDEFINED.

If [ERRIDR_EL1](#) indicates that zero error records are implemented, then it is IMPLEMENTATION DEFINED whether ERRSELR_EL1 is UNDEFINED or RES0.

Attributes

ERRSELR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | SEL | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

SEL, bits [15:0]

Selects the error record accessed through the ERX registers.

For example, if ERRSELR_EL1.SEL is 0x0004, then direct reads and writes of [ERXSTATUS_EL1](#) access ERR4STATUS.

If ERRSELR_EL1.SEL is greater than or equal to [ERRIDR_EL1.NUM](#), then all of the following apply:

- The value read back from ERRSELR_EL1.SEL is UNKNOWN.
- One of the following occurs:
 - An UNKNOWN error record is selected.
 - The ERX*_EL1 registers are RAZ/WI.
 - ERX*_EL1 register reads and writes are NOPs.
 - ERX*_EL1 register reads and writes are UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ERRSELR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERRSELR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ERRSELR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ERRSELR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ERRSELR_EL1;
    elsif PSTATE.EL == EL3 then
        return ERRSELR_EL1;

```

MSR ERRSELR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERRSELR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERRSELR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERRSELR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ERRSELR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXADDR_EL1, Selected Error Record Address Register

The ERXADDR_EL1 characteristics are:

Purpose

Accesses [ERR<n>ADDR](#) for the error record <n> selected by [ERRSELR_EL1](#).SEL.

Configuration

AArch64 System register ERXADDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ERXADDR\[31:0\]](#).

AArch64 System register ERXADDR_EL1 bits [63:32] are architecturally mapped to AArch32 System register [ERXADDR2\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXADDR_EL1 are UNDEFINED.

Attributes

ERXADDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ERR<n>ADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ERR<n>ADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

ERXADDR_EL1 accesses [ERR<n>ADDR](#), where <n> is the value in [ERRSELR_EL1](#).SEL.

Accessing ERXADDR_EL1

If [ERRIDR_EL1](#).NUM is 0x0000 or [ERRSELR_EL1](#).SEL is greater than or equal to [ERRIDR_EL1](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXADDR_EL1 is RAZ/WI.
- Direct reads and writes of ERXADDR_EL1 are NOPs.
- Direct reads and writes of ERXADDR_EL1 are UNDEFINED.

[ERR<n>ADDR](#) describes additional constraints that also apply when [ERR<n>ADDR](#) is accessed through ERXADDR_EL1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERXADDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ERXADDR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ERXADDR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ERXADDR_EL1;
    elsif PSTATE.EL == EL3 then
        return ERXADDR_EL1;

```

MSR ERXADDR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERXADDR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXADDR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXADDR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ERXADDR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXCTLR_EL1, Selected Error Record Control Register

The ERXCTLR_EL1 characteristics are:

Purpose

Accesses [ERR<n>CTLR](#) for the error record <n> selected by [ERRSELR_EL1](#).SEL.

Configuration

AArch64 System register ERXCTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ERXCTLR\[31:0\]](#).

AArch64 System register ERXCTLR_EL1 bits [63:32] are architecturally mapped to AArch32 System register [ERXCTLR2\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXCTLR_EL1 are UNDEFINED.

Attributes

ERXCTLR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ERR<n>CTLR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ERR<n>CTLR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

ERXCTLR_EL1 accesses [ERR<n>CTLR](#), where <n> is the value in [ERRSELR_EL1](#).SEL.

Accessing ERXCTLR_EL1

If [ERRIDR_EL1](#).NUM is 0x0000 or [ERRSELR_EL1](#).SEL is greater than or equal to [ERRIDR_EL1](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXCTLR_EL1 is RAZ/WI.
- Direct reads and writes of ERXCTLR_EL1 are NOPs.
- Direct reads and writes of ERXCTLR_EL1 are UNDEFINED.

If [ERRSELR_EL1](#).SEL is not the index of the first error record owned by a node, then [ERR<n>CTLR](#) is not present, meaning reads and writes of ERXCTLR_EL1 are RES0.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERXCTLR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elseif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGRTR_EL2.ERXCTLR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ERXCTLR_EL1;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ERXCTLR_EL1;
elseif PSTATE.EL == EL3 then
    return ERXCTLR_EL1;
    
```

MSR ERXCTLR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERXCTLR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXCTLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXCTLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ERXCTLR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXFR_EL1, Selected Error Record Feature Register

The ERXFR_EL1 characteristics are:

Purpose

Accesses [ERR<n>FR](#) for the error record <n> selected by [ERRSELR_EL1](#).SEL.

Configuration

AArch64 System register ERXFR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ERXFR\[31:0\]](#).

AArch64 System register ERXFR_EL1 bits [63:32] are architecturally mapped to AArch32 System register [ERXFR2\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXFR_EL1 are UNDEFINED.

Attributes

ERXFR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | ERR<n>FR | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | ERR<n>FR | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

ERXFR_EL1 accesses [ERR<n>FR](#), where <n> is the value in [ERRSELR_EL1](#).SEL.

Accessing ERXFR_EL1

If [ERRIDR_EL1](#).NUM is 0x0000 or [ERRSELR_EL1](#).SEL is greater than or equal to [ERRIDR_EL1](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXFR_EL1 is RAZ.
- Direct reads of ERXFR_EL1 are NOPs.
- Direct reads of ERXFR_EL1 are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERXFR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGRTR_EL2.ERXFR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ERXFR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ERXFR_EL1;
elsif PSTATE.EL == EL3 then
    return ERXFR_EL1;
    
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC0_EL1, Selected Error Record Miscellaneous Register 0

The ERXMISC0_EL1 characteristics are:

Purpose

Accesses [ERR<n>MISC0](#) for the error record <n> selected by [ERRSELR_EL1](#).SEL.

Configuration

AArch64 System register ERXMISC0_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ERXMISC0\[31:0\]](#).

AArch64 System register ERXMISC0_EL1 bits [63:32] are architecturally mapped to AArch32 System register [ERXMISC1\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXMISC0_EL1 are UNDEFINED.

Attributes

ERXMISC0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ERR<n>MISC0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ERR<n>MISC0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

ERXMISC0_EL1 accesses [ERR<n>MISC0](#), where <n> is the value in [ERRSELR_EL1](#).SEL.

Accessing ERXMISC0_EL1

If [ERRIDR_EL1](#).NUM is 0x0000 or [ERRSELR_EL1](#).SEL is greater than or equal to [ERRIDR_EL1](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC0_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC0_EL1 are NOPs.
- Direct reads and writes of ERXMISC0_EL1 are UNDEFINED.

[ERR<n>MISC0](#) describes additional constraints that also apply when [ERR<n>MISC0](#) is accessed through ERXMISC0_EL1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERXMISC0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ERXMISCn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ERXMISC0_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ERXMISC0_EL1;
    elsif PSTATE.EL == EL3 then
        return ERXMISC0_EL1;

```

MSR ERXMISC0_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERXMISCn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXMISC0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXMISC0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ERXMISC0_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC1_EL1, Selected Error Record Miscellaneous Register 1

The ERXMISC1_EL1 characteristics are:

Purpose

Accesses [ERR<n>MISC1](#) for the error record <n> selected by [ERRSELR_EL1](#).SEL.

Configuration

AArch64 System register ERXMISC1_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ERXMISC2\[31:0\]](#).

AArch64 System register ERXMISC1_EL1 bits [63:32] are architecturally mapped to AArch32 System register [ERXMISC3\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXMISC1_EL1 are UNDEFINED.

Attributes

ERXMISC1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ERR<n>MISC1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ERR<n>MISC1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

ERXMISC1_EL1 accesses [ERR<n>MISC1](#), where <n> is the value in [ERRSELR_EL1](#).SEL.

Accessing ERXMISC1_EL1

If [ERRIDR_EL1](#).NUM is 0x0000 or [ERRSELR_EL1](#).SEL is greater than or equal to [ERRIDR_EL1](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC1_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC1_EL1 are NOPs.
- Direct reads and writes of ERXMISC1_EL1 are UNDEFINED.

[ERR<n>MISC1](#) describes additional constraints that also apply when [ERR<n>MISC1](#) is accessed through ERXMISC1_EL1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERXMISC1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elseif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ERXMISCn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXMISC1_EL1;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXMISC1_EL1;
elseif PSTATE.EL == EL3 then
    return ERXMISC1_EL1;

```

MSR ERXMISC1_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERXMISCn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXMISC1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXMISC1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ERXMISC1_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC2_EL1, Selected Error Record Miscellaneous Register 2

The ERXMISC2_EL1 characteristics are:

Purpose

Accesses [ERR<n>MISC2](#) for the error record <n> selected by [ERRSELR_EL1](#).SEL.

Configuration

AArch64 System register ERXMISC2_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ERXMISC4\[31:0\]](#).

AArch64 System register ERXMISC2_EL1 bits [63:32] are architecturally mapped to AArch32 System register [ERXMISC5\[31:0\]](#).

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXMISC2_EL1 are UNDEFINED.

Attributes

ERXMISC2_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ERR<n>MISC2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ERR<n>MISC2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

ERXMISC2_EL1 accesses [ERR<n>MISC2](#), where <n> is the value in [ERRSELR_EL1](#).SEL.

Accessing ERXMISC2_EL1

If [ERRIDR_EL1](#).NUM is 0x0000 or [ERRSELR_EL1](#).SEL is greater than or equal to [ERRIDR_EL1](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC2_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC2_EL1 are NOPs.
- Direct reads and writes of ERXMISC2_EL1 are UNDEFINED.

[ERR<n>MISC2](#) describes additional constraints that also apply when [ERR<n>MISC2](#) is accessed through ERXMISC2_EL1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERXMISC2_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0101 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elseif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ERXMISCn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXMISC2_EL1;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXMISC2_EL1;
elseif PSTATE.EL == EL3 then
    return ERXMISC2_EL1;
    
```

MSR ERXMISC2_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0101 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERXMISCN_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXMISC2_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXMISC2_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ERXMISC2_EL1 = X[t];
    
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC3_EL1, Selected Error Record Miscellaneous Register 3

The ERXMISC3_EL1 characteristics are:

Purpose

Accesses [ERR<n>MISC3](#) for the error record <n> selected by [ERRSELR_EL1](#).SEL.

Configuration

AArch64 System register ERXMISC3_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ERXMISC6\[31:0\]](#).

AArch64 System register ERXMISC3_EL1 bits [63:32] are architecturally mapped to AArch32 System register [ERXMISC7\[31:0\]](#).

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXMISC3_EL1 are UNDEFINED.

Attributes

ERXMISC3_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ERR<n>MISC3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ERR<n>MISC3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

ERXMISC3_EL1 accesses [ERR<n>MISC3](#), where <n> is the value in [ERRSELR_EL1](#).SEL.

Accessing ERXMISC3_EL1

If [ERRIDR_EL1](#).NUM is 0x0000 or [ERRSELR_EL1](#).SEL is greater than or equal to [ERRIDR_EL1](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC3_EL1 is RAZ/WI.
- Direct reads and writes of ERXMISC3_EL1 are NOPs.
- Direct reads and writes of ERXMISC3_EL1 are UNDEFINED.

[ERR<n>MISC3](#) describes additional constraints that also apply when [ERR<n>MISC3](#) is accessed through ERXMISC3_EL1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERXMISC3_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0101 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ERXMISCn_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ERXMISC3_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ERXMISC3_EL1;
    elsif PSTATE.EL == EL3 then
        return ERXMISC3_EL1;

```

MSR ERXMISC3_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0101 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERXMISCN_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXMISC3_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXMISC3_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ERXMISC3_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXPFGCDN_EL1, Selected Pseudo-fault Generation Countdown register

The ERXPFGCDN_EL1 characteristics are:

Purpose

Accesses [ERR<n>PFGCDN](#) for the error record <n> selected by [ERRSELR_EL1](#).SEL.

Configuration

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXPFGCDN_EL1 are UNDEFINED.

Attributes

ERXPFGCDN_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ERR<n>PFGCDN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

ERXPFGCDN_EL1 accesses [ERR<n>PFGCDN](#), where <n> is the value in [ERRSELR_EL1](#).SEL.

Accessing ERXPFGCDN_EL1

If [ERRIDR_EL1](#).NUM is 0x0000 or [ERRSELR_EL1](#).SEL is greater than or equal to [ERRIDR_EL1](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXPFGCDN_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCDN_EL1 are NOPs.
- Direct reads and writes of ERXPFGCDN_EL1 are UNDEFINED.

If [ERRSELR_EL1](#).SEL selects an error record owned by a node that does not implement the Common Fault Injection Model Extension, then one of the following occurs:

- ERXPFGCDN_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCDN_EL1 are NOPs.
- Direct reads and writes of ERXPFGCDN_EL1 are UNDEFINED.

Note

A node does not implement the Common Fault Injection Model Extension if [ERR<q>FR](#).INJ reads as 0b00. <q> is the index of the first error record owned by the same node as error record <n>, where <n> is the value in [ERRSELR_EL1](#).SEL. If the node owns a single record, then q = n.

If [ERRSELR_EL1](#).SEL is not the index of the first error record owned by a node, then [ERR<n>PFGCDN](#) is not present, meaning reads and writes of ERXPFGCDN_EL1 are RES0.

[ERR<n>PFGCDN](#) describes additional constraints that also apply when [ERR<n>PFGCDN](#) is accessed through ERXPFPGCDN_EL1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERXPFPGCDN_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ERXPFPGCDN_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXPFPGCDN_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXPFPGCDN_EL1;
elsif PSTATE.EL == EL3 then
    return ERXPFPGCDN_EL1;

```

MSR ERXPFPGCDN_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERXPFGCDN_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXPFGCDN_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXPFGCDN_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ERXPFGCDN_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXPFGCTL_EL1, Selected Pseudo-fault Generation Control register

The ERXPFGCTL_EL1 characteristics are:

Purpose

Accesses [ERR<n>PFGCTL](#) for the error record <n> selected by [ERRSELR_EL1](#).SEL.

Configuration

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXPFGCTL_EL1 are UNDEFINED.

Attributes

ERXPFGCTL_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ERR<n>PFGCTL | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ERR<n>PFGCTL | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:0]

ERXPFGCTL_EL1 accesses [ERR<n>PFGCTL](#), where <n> is the value in [ERRSELR_EL1](#).SEL.

Accessing ERXPFGCTL_EL1

If [ERRIDR_EL1](#).NUM is 0x0000 or [ERRSELR_EL1](#).SEL is greater than or equal to [ERRIDR_EL1](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXPFGCTL_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCTL_EL1 are NOPs.
- Direct reads and writes of ERXPFGCTL_EL1 are UNDEFINED.

If [ERRSELR_EL1](#).SEL selects an error record owned by a node that does not implement the Common Fault Injection Model Extension, then one of the following occurs:

- ERXPFGCTL_EL1 is RAZ/WI.
- Direct reads and writes of ERXPFGCTL_EL1 are NOPs.
- Direct reads and writes of ERXPFGCTL_EL1 are UNDEFINED.

Note

A node does not implement the Common Fault Injection Model Extension if [ERR<q>FR](#).INJ reads as 0b00. <q> is the index of the first error record owned by the same node as error record <n>, where <n> is the value in [ERRSELR_EL1](#).SEL. If the node owns a single record, then q = n.

If [ERRSELR_EL1](#).SEL is not the index of the first error record owned by a node, then [ERR<n>PFGCTL](#) is not present, meaning reads and writes of ERXPFGCTL_EL1 are RES0.

[ERR<n>PFGCTL](#) describes additional constraints that also apply when [ERR<n>PFGCTL](#) is accessed through ERXPFPGCTL_EL1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERXPFPGCTL_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ERXPFPGCTL_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXPFPGCTL_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXPFPGCTL_EL1;
elsif PSTATE.EL == EL3 then
    return ERXPFPGCTL_EL1;

```

MSR ERXPFPGCTL_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERXPFPGCTL_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXPFPGCTL_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXPFPGCTL_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ERXPFPGCTL_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXPFGF_EL1, Selected Pseudo-fault Generation Feature register

The ERXPFGF_EL1 characteristics are:

Purpose

Accesses [ERR<n>PFGF](#) for the error record <n> selected by [ERRSELR_EL1](#).SEL.

Configuration

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXPFGF_EL1 are UNDEFINED.

Attributes

ERXPFGF_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ERR<n>PFGF | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

ERXPFGF_EL1 accesses [ERR<n>PFGF](#), where <n> is the value in [ERRSELR_EL1](#).SEL.

Accessing ERXPFGF_EL1

If [ERRIDR_EL1](#).NUM is 0x0000 or [ERRSELR_EL1](#).SEL is greater than or equal to [ERRIDR_EL1](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXPFGF_EL1 is RAZ.
- Direct reads of ERXPFGF_EL1 are NOPs.
- Direct reads of ERXPFGF_EL1 are UNDEFINED.

If [ERRSELR_EL1](#).SEL selects an error record owned by a node that does not implement the Common Fault Injection Model Extension, then one of the following occurs:

- ERXPFGF_EL1 is RAZ.
- Direct reads of ERXPFGF_EL1 are NOPs.
- Direct reads of ERXPFGF_EL1 are UNDEFINED.

Note

A node does not implement the Common Fault Injection Model Extension if [ERR<q>FR](#).INJ reads as 0b00. <q> is the index of the first error record owned by the same node as error record <n>, where <n> is the value in [ERRSELR_EL1](#).SEL. If the node owns a single record, then q = n.

If [ERRSELR_EL1](#).SEL is not the index of the first error record owned by a node, then [ERR<n>PFGF](#) is not present, meaning reads of ERXPFGF_EL1 are RES0.

[ERR<n>PFGF](#) describes additional constraints that also apply when [ERR<n>PFGF](#) is accessed through ERXPFPGF_EL1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERXPFPGF_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.FIEN == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ERXPFPGF_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ERXPFPGF_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIEN == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FIEN == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ERXPFPGF_EL1;
elsif PSTATE.EL == EL3 then
    return ERXPFPGF_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXSTATUS_EL1, Selected Error Record Primary Status Register

The ERXSTATUS_EL1 characteristics are:

Purpose

Accesses [ERR<n>STATUS](#) for the error record <n> selected by [ERRSELR_EL1](#).SEL.

Configuration

AArch64 System register ERXSTATUS_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ERXSTATUS\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXSTATUS_EL1 are UNDEFINED.

Attributes

ERXSTATUS_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ERR<n>STATUS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ERR<n>STATUS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

ERXSTATUS_EL1 accesses [ERR<n>STATUS](#), where <n> is the value in [ERRSELR_EL1](#).SEL.

Accessing ERXSTATUS_EL1

If [ERRIDR_EL1](#).NUM is 0x0000 or [ERRSELR_EL1](#).SEL is greater than or equal to [ERRIDR_EL1](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXSTATUS_EL1 is RAZ/WI.
- Direct reads and writes of ERXSTATUS_EL1 are NOPs.
- Direct reads and writes of ERXSTATUS_EL1 are UNDEFINED.

[ERR<n>STATUS](#) describes additional constraints that also apply when [ERR<n>STATUS](#) is accessed through ERXSTATUS_EL1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ERXSTATUS_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elseif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ERXSTATUS_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXSTATUS_EL1;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ERXSTATUS_EL1;
elseif PSTATE.EL == EL3 then
    return ERXSTATUS_EL1;
    
```

MSR ERXSTATUS_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TERR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ERXSTATUS_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXSTATUS_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ERXSTATUS_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ERXSTATUS_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ESR_EL1, Exception Syndrome Register (EL1)

The ESR_EL1 characteristics are:

Purpose

Holds syndrome information for an exception taken to EL1.

Configuration

AArch64 System register ESR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DFSR\[31:0\]](#).

Attributes

ESR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ISS2 | | | |
| EC | | | | | | IL | | ISS | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

ESR_EL1 is made UNKNOWN as a result of an exception return from EL1.

When an UNPREDICTABLE instruction is treated as UNDEFINED, and the exception is taken to EL1, the value of ESR_EL1 is UNKNOWN. The value written to ESR_EL1 must be consistent with a value that could be created as a result of an exception from the same Exception level that generated the exception as a result of a situation that is not UNPREDICTABLE at that Exception level, in order to avoid the possibility of a privilege violation.

Bits [63:37]

Reserved, RES0.

ISS2, bits [36:32]

When FEAT_LS64 is implemented:

If a memory access generated by an ST64BV or ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field holds register specifier, Xs.

For any other Data Abort, this field is RES0.

Otherwise:

Reserved, RES0.

EC, bits [31:26]

Exception Class. Indicates the reason for the exception that this register holds information about.

For each EC value, the table references a subsection that gives information about:

- The cause of the exception, for example the configuration required to enable the trap.
- The encoding of the associated ISS.

Possible values of the EC field are:

| EC | Meaning | ISS | Applies when |
|----------|---|--|---------------------------|
| 0b000000 | Unknown reason. | ISS encoding for exceptions with an unknown reason | |
| 0b000001 | Trapped WF* instruction execution. Conditional WF* instructions that fail their condition code check do not cause an exception. | ISS encoding for an exception from a WF* instruction | |
| 0b000011 | Trapped MCR or MRC access with (coproc==0b1111) that is not reported using EC 0b000000. | ISS encoding for an exception from an MCR or MRC access | When AArch32 is supported |
| 0b000100 | Trapped MCRR or MRRC access with (coproc==0b1111) that is not reported using EC 0b000000. | ISS encoding for an exception from an MCRR or MRRC access | When AArch32 is supported |
| 0b000101 | Trapped MCR or MRC access with (coproc==0b1110). | ISS encoding for an exception from an MCR or MRC access | When AArch32 is supported |
| 0b000110 | Trapped LDC or STC access. The only architected uses of these instruction are: <ul style="list-style-type: none"> An STC to write data to memory from DBGDTRRXint. An LDC to read data from memory to DBGDTRTXint. | ISS encoding for an exception from an LDC or STC instruction | When AArch32 is supported |
| 0b000111 | Access to SME, SVE, Advanced SIMD or floating-point functionality trapped by CPACR_EL1.FPEN , CPTR_EL2.FPEN , CPTR_EL2.TFP , or CPTR_EL3.TFP control. Excludes exceptions resulting from CPACR_EL1 when the value of HCR_EL2.TGE is 1, or because SVE or Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000 as described in 'The EC used to report an exception routed to EL2 because HCR_EL2.TGE is 1'. | ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality resulting from the FPEN and TFP traps | |
| 0b001010 | Trapped execution of an LD64B, | ISS encoding for an exception | When FEAT_LS64 |

| | | | |
|----------|---|--|------------------------------|
| | ST64B, ST64BV, or ST64BV0 instruction. | from an LD64B or ST64B* instruction | is implemented |
| 0b001100 | Trapped MRRC access with (coproc==0b1110). | ISS encoding for an exception from an MCRR or MRRC access | When AArch32 is supported |
| 0b001101 | Branch Target Exception. | ISS encoding for an exception from Branch Target Identification instruction | When FEAT_BTI is implemented |
| 0b001110 | Illegal Execution state. | ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault | |
| 0b010001 | SVC instruction execution in AArch32 state. | ISS encoding for an exception from HVC or SVC instruction execution | When AArch32 is supported |
| 0b010101 | SVC instruction execution in AArch64 state. | ISS encoding for an exception from HVC or SVC instruction execution | When AArch64 is supported |
| 0b011000 | Trapped MSR, MRS or System instruction execution in AArch64 state, that is not reported using EC 0b000000, 0b000001, or 0b000111. This includes all instructions that cause exceptions that are part of the encoding space defined in 'System instruction class encoding overview', except for those exceptions reported using EC values 0b000000, 0b000001, or 0b000111. | ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state | When AArch64 is supported |
| 0b011001 | Access to SVE functionality trapped as a result of CPACR_EL1.ZEN , CPTR_EL2.ZEN , CPTR_EL2.TZ , or CPTR_EL3.EZ , that is not reported using EC 0b000000. | ISS encoding for an exception from an access to SVE functionality resulting from CPACR_EL1.ZEN, CPTER_EL2.ZEN, CPTER_EL2.TZ, or CPTER_EL3.EZ | When FEAT_SVE is implemented |
| 0b011011 | Exception from an access to a TSTART instruction at EL0 when SCTLR_EL1.TME0 == 0, EL0 when SCTLR_EL2.TME0 == 0, at EL1 when SCTLR_EL1.TME | ISS encoding for an exception from a TSTART instruction | When FEAT_TME is implemented |

| | | | |
|----------|---|--|-------------------------------|
| | <p>== 0, at EL2 when SCTLR_EL2.TME == 0 or at EL3 when SCTLR_EL3.TME == 0.</p> | | |
| 0b011100 | Exception from a Pointer Authentication instruction authentication failure | ISS encoding for an exception from a Pointer Authentication instruction authentication failure | When FEAT_FPAC is implemented |
| 0b011101 | Access to SME functionality trapped as a result of CPACR_EL1.SMEN , CPTR_EL2.SMEN , CPTR_EL2.TSM , CPTR_EL3.ESM , or an attempted execution of an instruction that is illegal because of the value of PSTATE.SM or PSTATE.ZA, that is not reported using EC 0b000000. | ISS encoding for an exception due to SME functionality | When FEAT_SME is implemented |
| 0b011110 | Exception from a Granule Protection Check | ISS encoding for an exception from a Granule Protection Check | When FEAT_RME is implemented |
| 0b100000 | Instruction Abort from a lower Exception level. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions. | ISS encoding for an exception from an Instruction Abort | |
| 0b100001 | Instruction Abort taken without a change in Exception level. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions. | ISS encoding for an exception from an Instruction Abort | |
| 0b100010 | PC alignment fault exception. | ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault | |
| 0b100100 | Data Abort from a lower Exception level. | ISS encoding for an exception | |

| | | | |
|----------|---|--|-------------------------------|
| | Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions. | from a Data Abort | |
| 0b100101 | Data Abort taken without a change in Exception level. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions. | ISS encoding for an exception from a Data Abort | |
| 0b100110 | SP alignment fault exception. | ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault | |
| 0b100111 | Memory Operation Exception. | ISS encoding for an exception from the Memory Copy and Memory Set instructions | When FEAT_MOPS is implemented |
| 0b101000 | Trapped floating-point exception taken from AArch32 state. This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED. | ISS encoding for an exception from a trapped floating-point exception | When AArch32 is supported |
| 0b101100 | Trapped floating-point exception taken from AArch64 state. This EC value is valid if the | ISS encoding for an exception from a trapped floating-point exception | When AArch64 is supported |

| | | | |
|----------|--|---|---------------------------|
| | implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED. | | |
| 0b101111 | SError interrupt. | ISS encoding for an SError interrupt | |
| 0b110000 | Breakpoint exception from a lower Exception level. | ISS encoding for an exception from a Breakpoint or Vector Catch debug exception | |
| 0b110001 | Breakpoint exception taken without a change in Exception level. | ISS encoding for an exception from a Breakpoint or Vector Catch debug exception | |
| 0b110010 | Software Step exception from a lower Exception level. | ISS encoding for an exception from a Software Step exception | |
| 0b110011 | Software Step exception taken without a change in Exception level. | ISS encoding for an exception from a Software Step exception | |
| 0b110100 | Watchpoint exception from a lower Exception level. | ISS encoding for an exception from a Watchpoint exception | |
| 0b110101 | Watchpoint exception taken without a change in Exception level. | ISS encoding for an exception from a Watchpoint exception | |
| 0b111000 | BKPT instruction execution in AArch32 state. | ISS encoding for an exception from execution of a Breakpoint instruction | When AArch32 is supported |
| 0b111100 | BRK instruction execution in AArch64 state. | ISS encoding for an exception from execution of a Breakpoint instruction | When AArch64 is supported |

All other EC values are reserved by Arm, and:

- Unused values in the range 0b000000 - 0b101100 (0x00 - 0x2C) are reserved for future use for synchronous exceptions.
- Unused values in the range 0b101101 - 0b111111 (0x2D - 0x3F) are reserved for future use, and might be used for synchronous or asynchronous exceptions.

The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [25]

Instruction Length for synchronous exceptions. Possible values of this bit are:

| IL | Meaning |
|-----|--|
| 0b0 | 16-bit instruction trapped. |
| 0b1 | 32-bit instruction trapped. This value is also used when the exception is one of the following: <ul style="list-style-type: none"> • An SError interrupt. • An Instruction Abort exception. • A PC alignment fault exception. • An SP alignment fault exception. • A Data Abort exception for which the value of the ISV bit is 0. • An Illegal Execution state exception. • Any debug exception except for Breakpoint instruction exceptions. For Breakpoint instruction exceptions, this bit has its standard meaning: <ul style="list-style-type: none"> ◦ 0b0: 16-bit T32 BKPT instruction. ◦ 0b1: 32-bit A32 BKPT instruction or A64 BRK instruction. • An exception reported using EC value 0b000000. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [24:0]

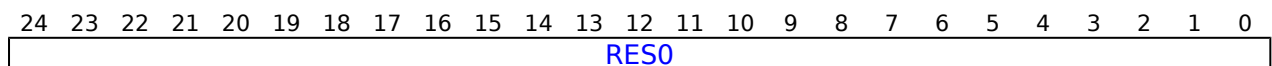
Instruction Specific Syndrome. Architecturally, this field can be defined independently for each defined Exception class. However, in practice, some ISS encodings are used for more than one Exception class.

Typically, an ISS encoding has a number of subfields. When an ISS subfield holds a register number, the value returned in that field is the AArch64 view of the register number.

For an exception taken from AArch32 state, see 'Mapping of the general-purpose registers between the Execution states'.

If the AArch32 register descriptor is 0b1111, then:

- If the instruction that generated the exception was not UNPREDICTABLE, the field takes the value 0b11111.
- If the instruction that generated the exception was UNPREDICTABLE, the field takes an UNKNOWN value that must be either:
 - The AArch64 view of the register number of a register that might have been used at the Exception level from which the exception was taken.
 - The value 0b11111.

ISS encoding for exceptions with an unknown reason**Bits [24:0]**

Reserved, RES0.

When an exception is reported using this EC code the IL field is set to 1.

This EC code is used for all exceptions that are not covered by any other EC value. This includes exceptions that are generated in the following situations:

- The attempted execution of an instruction bit pattern that has no allocated instruction or that is not accessible at the current Exception level and Security state, including:
 - A read access using a System register pattern that is not allocated for reads or that does not permit reads at the current Exception level and Security state.

- A write access using a System register pattern that is not allocated for writes or that does not permit writes at the current Exception level and Security state.
- Instruction encodings that are unallocated.
- Instruction encodings for instructions or System registers that are not implemented in the implementation.
- In Debug state, the attempted execution of an instruction bit pattern that is not accessible in Debug state.
- In Non-debug state, the attempted execution of an instruction bit pattern that is not accessible in Non-debug state.
- In AArch32 state, attempted execution of a short vector floating-point instruction.
- In an implementation that does not include Advanced SIMD and floating-point functionality, an attempted access to Advanced SIMD or floating-point functionality under conditions where that access would be permitted if that functionality was present. This includes the attempted execution of an Advanced SIMD or floating-point instruction, and attempted accesses to Advanced SIMD and floating-point System registers.
- An exception generated because of the value of one of the [SCTLR_EL1](#).{ITD, SED, CP15BEN} control bits.
- Attempted execution of:
 - An HVC instruction when disabled by [HCR_EL2](#).HCD or [SCR_EL3](#).HCE.
 - An SMC instruction when disabled by [SCR_EL3](#).SMD.
 - An HLT instruction when disabled by [EDSCR](#).HDE.
- Attempted execution of an MSR or MRS instruction to access [SP_EL0](#) when the value of [SPSel](#).SP is 0.
- Attempted execution of an MSR or MRS instruction using a [_EL12](#) register name when [HCR_EL2](#).E2H == 0.
- Attempted execution, in Debug state, of:
 - A DCPS1 instruction when the value of [HCR_EL2](#).TGE is 1 and EL2 is disabled or not implemented in the current Security state.
 - A DCPS2 instruction from EL1 or EL0 when EL2 is disabled or not implemented in the current Security state.
 - A DCPS3 instruction when the value of [EDSCR](#).SDD is 1, or when EL3 is not implemented.
- When EL3 is using AArch64, attempted execution from Secure EL1 of an SRS instruction using R13_mon. See 'Traps to EL3 of Secure monitor functionality from Secure EL1 using AArch32'.
- In Debug state when the value of [EDSCR](#).SDD is 1, the attempted execution at EL2, EL1, or EL0 of an instruction that is configured to trap to EL3.
- In AArch32 state, the attempted execution of an MRS (banked register) or an MSR (banked register) instruction to [SPSR_mon](#), [SP_mon](#), or [LR_mon](#).
- An exception that is taken to EL2 because the value of [HCR_EL2](#).TGE is 1 that, if the value of [HCR_EL2](#).TGE was 0 would have been reported with an [ESR_ELx](#).EC value of 0b000111.
- In Non-transactional state, attempted execution of a TCOMMIT instruction.

ISS encoding for an exception from a WF* instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|----|----|----|----|----|----|----|---|---|------|---|----|----|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | RES0 | | | | | | | | | | RN | | | RES0 | | RV | TI | | | |

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:10]

Reserved, RES0.

RN, bits [9:5]

When FEAT_WFxT is implemented:

Register Number. Indicates the register number supplied for a WFET or WFIT instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [4:3]

Reserved, RES0.

RV, bit [2]

When FEAT_WFxT is implemented:

Register field Valid.

If TI[1] == 1, then this field indicates whether RN holds a valid register number for the register argument to the trapped WFET or WFIT instruction.

| RV | Meaning |
|-----|-------------------------|
| 0b0 | Register field invalid. |
| 0b1 | Register field valid. |

If TI[1] == 0, then this field is RES0.

This field is set to 1 on a trap on WFET or WFIT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TI, bits [1:0]

Trapped instruction. Possible values of this bit are:

| TI | Meaning | Applies when |
|------|---------------|-------------------------------|
| 0b00 | WFI trapped. | |
| 0b01 | WFE trapped. | |
| 0b10 | WFIT trapped. | When FEAT_WFxF is implemented |
| 0b11 | WFET trapped. | When FEAT_WFxF is implemented |

When FEAT_WFxF is implemented, this is a two bit field as shown. Otherwise, bit[1] is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following fields describe configuration settings for generating this exception:

- [SCTLR_EL1](#).{nTWE, nTWI}.
- [HCR_EL2](#).{TWE, TWI}.
- [SCR_EL3](#).{TWE, TWI}.

ISS encoding for an exception from an MCR or MRC access

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|------|----|----|-----|----|----|----|---|---|-----|---|---|-----------|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | Opc2 | | | Opc1 | | | CRn | | | Rt | | | CRm | | | Direction | | | | |

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc2, bits [19:17]

The Opc2 value from the issued instruction.

For a trapped VMRS access, holds the value 0b000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [16:14]

The Opc1 value from the issued instruction.

For a trapped VMRS access, holds the value 0b111.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

For a trapped VMRS access, holds the reg field from the VMRS instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

If the Rt value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b1111.

- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
 - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
 - The value 0b11111.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

For a trapped VMRS access, holds the value 0b0000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|---|
| 0b0 | Write to System register space. MCR instruction. |
| 0b1 | Read from System register space. MRC or VMRS instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000011:

- [CNTCTL_EL1](#).{ELOPTEN, EL0VTEN, ELOPCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [PMUSERENR_EL0](#).{ER, CR, SW, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [AMUSERENR_EL0](#).EN, for accesses to Activity Monitors registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [HCR_EL2](#).{TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).TTLB, for execution of TLB maintenance instructions at EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).{TSW, TPC, TPU} for execution of cache maintenance instructions at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).TACR, for accesses to the Auxiliary Control Register at EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).TIDCP, for accesses to lockdown, DMA, and TCM operations at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).{TID1, TID2, TID3}, for accesses to ID registers at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [CPTR_EL2](#).TCPAC, for accesses to [CPACR_EL1](#) or [CPACR](#) using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HSTR_EL2](#).T<n>, for accesses to System registers using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [CNTHCTL_EL2](#).EL1PCEN, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [MDCR_EL2](#).{TPM, TPMCR}, for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.

- [CPTR_EL2](#).TAM, for accesses to Activity Monitors registers from EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [CPTR_EL3](#).TCPAC, for accesses to [CPACR](#) from EL1 and EL2, and accesses to [HCPTR](#) from EL2 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL3.
- [MDCR_EL3](#).TPM, for accesses to Performance Monitor registers from EL0, EL1 and EL2 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL3.
- [CPTR_EL3](#).TAM, for accesses to Activity Monitors registers from EL0, EL1 and EL2 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL3.
- For information on other traps using EC value 0b000011, see 'Traps to EL3 of Secure monitor functionality from Secure EL1 using AArch32'.
- If FEAT_FGT is implemented, MCR or MRC access to some registers at EL0, trapped to EL2.

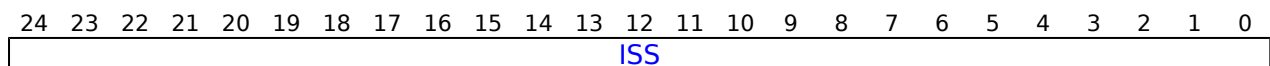
The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000101:

- [CPACR_EL1](#).TTA for accesses to trace registers, MCR or MRC access (coproc == 0b1110) trapped to EL1 or EL2.
- [MDSCR_EL1](#).TDCC, for accesses to the Debug Communications Channel (DCC) registers at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1110) trapped to EL1 or EL2.
- If FEAT_FGT is implemented, [MDCR_EL2](#).TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and [MDCR_EL3](#).TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.
- [HCR_EL2](#).TID0, for accesses to the [JIDR](#) register in the ID group 0 at EL0 and EL1 using AArch32, MRC access (coproc == 0b1110) trapped to EL2.
- [CPTR_EL2](#).TTA, for accesses to trace registers using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- [MDCR_EL2](#).TDRA, for accesses to Debug ROM registers [DBGDRAR](#) and [DBGDSAR](#) using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- [MDCR_EL2](#).TDOSA, for accesses to powerdown debug registers, using AArch32 state, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- [MDCR_EL2](#).TDA, for accesses to other debug registers, using AArch32 state, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- [CPTR_EL3](#).TTA, for accesses to trace registers using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL3.
- [MDCR_EL3](#).TDOSA, for accesses to powerdown debug registers using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL3.
- [MDCR_EL3](#).TDA, for accesses to other debug registers, using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL3.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b001000:

- [HCR_EL2](#).TID0, for accesses to the [FPSID](#) register in ID group 0 at EL1 using AArch32 state, VMRS access trapped to EL2.
- [HCR_EL2](#).TID3, for accesses to registers in ID group 3 including [MVFR0](#), [MVFR1](#) and [MVFR2](#), VMRS access trapped to EL2.

ISS encoding for an exception from an LD64B or ST64B* instruction



ISS, bits [24:0]

| ISS | Meaning |
|------------------------------------|-------------------------------------|
| 0b00000000000000000000000000000000 | ST64BV instruction trapped. |
| 0b00000000000000000000000000000001 | ST64BV0 instruction trapped. |
| 0b00000000000000000000000000000010 | LD64B or ST64B instruction trapped. |

All other values are reserved.

ISS encoding for an exception from an MCRR or MRRC access

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|------|-----|----|----|----|----|---|---|---|-----|---|---|---|-----------|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | Opc1 | | | | RES0 | Rt2 | | | | Rt | | | | CRm | | | | Direction | | |

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [19:16]

The Opc1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

Rt2, bits [14:10]

The Rt2 value from the issued instruction, the second general-purpose register used for the transfer.

If the Rt2 value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt2 value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b11111.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
 - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
 - The value 0b11111.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the first general-purpose register used for the transfer.

If the Rt value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b11111.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
 - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
 - The value 0b11111.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|--|
| 0b0 | Write to System register space. MCRR instruction. |
| 0b1 | Read from System register space. MRRC instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000100:

- [CNTKCTL_EL1](#).{ELOPTEN, EL0VTEN, ELOPCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [PMUSERENR_EL0](#).{CR, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [AMUSERENR_EL0](#).{EN}, for accesses to Activity Monitors registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [HCR_EL2](#).{TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [HSTR_EL2](#).T<n>, for accesses to System registers using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [CNTHCTL_EL2](#).{EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [MDCR_EL2](#).{TPM, TPMCR}, for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [CPTR_EL2](#).TAM, for accesses to Activity Monitors registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [MDCR_EL3](#).TPM, for accesses to Performance Monitor registers from EL0, EL1 and EL2 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL3.
- [CPTR_EL3](#).TAM, for accesses to Activity Monitors registers from EL0, EL1 and EL2 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL3.
- If FEAT_FGT is implemented, [HDFGRTR_EL2](#).PMCCNTR_EL0 for MRRC access and [HDFGWTR_EL2](#).PMCCNTR_EL0 for MCRR access to [PMCCNTR](#) at EL0, trapped to EL2.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b001100:

- [MDSCR_EL1](#).TDCC, for accesses to the Debug ROM registers [DBGDSAR](#) and [DBGDRAR](#) at EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1110) trapped to EL1 or EL2.
- [MDCR_EL2](#).TDRA, for accesses to Debug ROM registers [DBGDRAR](#) and [DBGDSAR](#) using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL2.
- [MDCR_EL3](#).TDA, for accesses to debug registers, using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL3.
- [CPACR_EL1](#).TTA for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL1 or EL2.
- [CPTR_EL2](#).TTA, for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL2.
- [CPTR_EL3](#).TTA, for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL3.

Note

If the Armv8-A architecture is implemented with an ETMv4 implementation, MCRR and MRRC accesses to trace registers are UNDEFINED and the resulting exception is higher priority than an exception due to these traps.

ISS encoding for an exception from an LDC or STC instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|----|----|----|----|------|----|---|---|---|--------|----|---|-----------|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | imm8 | | | | | | | | RES0 | Rn | | | | Offset | AM | | Direction | | | |

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

imm8, bits [19:12]

The immediate value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

Rn, bits [9:5]

The Rn value from the issued instruction, the general-purpose register used for the transfer.

If the Rn value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rn value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b11111.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
 - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
 - The value 0b11111.

See 'Mapping of the general-purpose registers between the Execution states'.

This field is valid only when AM[2] is 0, indicating an immediate form of the LDC or STC instruction. When AM[2] is 1, indicating a literal form of the LDC or STC instruction, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Offset, bit [4]

Indicates whether the offset is added or subtracted:

| Offset | Meaning |
|--------|------------------|
| 0b0 | Subtract offset. |
| 0b1 | Add offset. |

This bit corresponds to the U bit in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AM, bits [3:1]

Addressing mode. The permitted values of this field are:

| AM | Meaning |
|-------|---|
| 0b000 | Immediate unindexed. |
| 0b001 | Immediate post-indexed. |
| 0b010 | Immediate offset. |
| 0b011 | Immediate pre-indexed. |
| 0b100 | For a trapped STC instruction or a trapped T32 LDC instruction this encoding is reserved. |
| 0b110 | For a trapped STC instruction, this encoding is reserved. |

The values 0b101 and 0b111 are reserved. The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in System and memory-mapped registers and translation table entries'.

Bit [2] in this subfield indicates the instruction form, immediate or literal.

Bits [1:0] in this subfield correspond to the bits {P, W} in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|------------------------------------|
| 0b0 | Write to memory. STC instruction. |
| 0b1 | Read from memory. LDC instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following fields describe the configuration settings for the traps that are reported using EC value 0b000110:

- [MDSCR_EL1.TDCC](#), for accesses using AArch32 state, LDC access to [DBGDTRTXint](#) or STC access to [DBGDTRRXint](#) trapped to EL1 or EL2.
- [MDCR_EL2.TDA](#), for accesses using AArch32 state, LDC access to [DBGDTRTXint](#) or STC access to [DBGDTRRXint](#) MCR or MRC access trapped to EL2.
- [MDCR_EL3.TDA](#), for accesses using AArch32 state, LDC access to [DBGDTRTXint](#) or STC access to [DBGDTRRXint](#) MCR or MRC access trapped to EL3.
- If FEAT_FGT is implemented, [MDCR_EL2.TDCC](#) for LDC and STC accesses to the DCC registers at EL0 and EL1 trapped to EL2, and [MDCR_EL3.TDCC](#) for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.

ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from the FPen and TFP traps

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | RES0 | | | | | | | | | | | | | | | | | | | |

The accesses covered by this trap include:

- Execution of SVE or Advanced SIMD and floating-point instructions.
- Accesses to the Advanced SIMD and floating-point System registers.
- Execution of SME instructions.

For an implementation that does not include either SVE or support for Advanced SIMD and floating-point, the exception is reported using the EC value 0b000000.

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

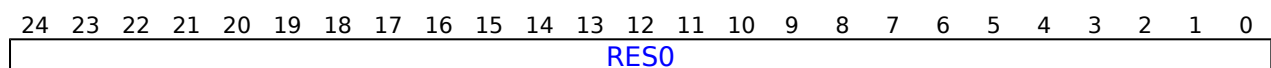
Bits [19:0]

Reserved, RES0.

The following fields describe the configuration settings for the traps that are reported using EC value 0b000111:

- [CPACR_EL1.FPEN](#), for accesses to SIMD and floating-point registers trapped to EL1.
- [CPTR_EL2.FPEN](#) and [CPTR_EL2.TFP](#), for accesses to SIMD and floating-point registers trapped to EL2.
- [CPTR_EL3.TFP](#), for accesses to SIMD and floating-point registers trapped to EL3.

ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ



The accesses covered by this trap include:

- Execution of SVE instructions when the PE is not in Streaming SVE mode.
- Accesses to the SVE System registers, ZCR_ELx.

For an implementation that does not include SVE, the exception is reported using the EC value 0b000000.

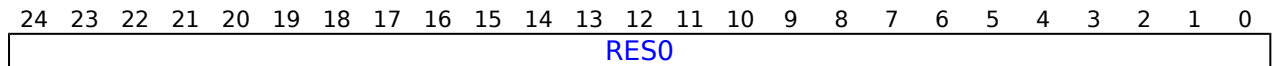
Bits [24:0]

Reserved, RES0.

The following fields describe the configuration settings for the traps that are reported using EC value 0b011001:

- [CPACR_EL1.ZEN](#), for execution of SVE instructions and accesses to SVE registers at EL0 or EL1, trapped to EL1.
- [CPTR_EL2.ZEN](#) and [CPTR_EL2.TZ](#), for execution of SVE instructions and accesses to SVE registers at EL0, EL1, or EL2, trapped to EL2.
- [CPTR_EL3.EZ](#), for execution of SVE instructions and accesses to SVE registers from all Exception levels, trapped to EL3.

ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault



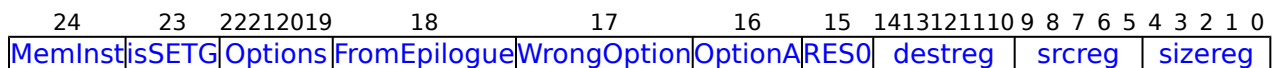
Bits [24:0]

Reserved, RES0.

There are no configuration settings for generating Illegal Execution state exceptions and PC alignment fault exceptions. For more information about these exceptions, see 'The Illegal Execution state exception' and 'PC alignment checking'.

'SP alignment checking' describes the configuration settings for generating SP alignment fault exceptions.

ISS encoding for an exception from the Memory Copy and Memory Set instructions



MemInst, bit [24]

Indicates the memory instruction class causing the exception.

| MemInst | Meaning |
|---------|--|
| 0b0 | CPYFE*, CPYFM*, CPYE*, and CPYM* instructions. |
| 0b1 | SETE*, SETM*, SETGE*, and SETGM* instructions. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

isSETG, bit [23]

Indicates whether the instruction belongs to SETGM* or SETGE* class of instruction.

| isSETG | Meaning |
|--------|-------------------------------------|
| 0b0 | Not a SETGM* or SETGE* instruction. |
| 0b1 | SETGM* or SETGE* instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Options, bits [22:19]

Options : the Options field of the instruction.

For Memory Copy instructions, bits[22:19] forms the Options field, which holds the bits[15:12] of the instruction.

For Memory Set instructions:

- Bits[22:21] are RES0.
- Bits[20:19] form the Options field, which holds the bits[13:12] of the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FromEpilogue, bit [18]

Indicates whether the instruction belongs to the epilogue class of Memory Copy or Memory Set instructions.

| FromEpilogue | Meaning |
|---------------------|--|
| 0b0 | Not an epilogue instruction. |
| 0b1 | CPYE*, CPYFE*, SETE*, or SETGE* instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WrongOption, bit [17]

Algorithm option.

| WrongOption | Meaning |
|--------------------|-----------------------|
| 0b0 | WrongOption is false. |
| 0b1 | WrongOption is true. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OptionA, bit [16]

Algorithm type indicated by the PSTATE.C bit.

| OptionA | Meaning |
|----------------|-------------------------------------|
| 0b0 | OptionB indicated by PSTATE.C is 0. |
| 0b1 | OptionA indicated by PSTATE.C is 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

destreg, bits [14:10]

The destination register value from the issued instruction, containing the destination address.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

srcreg, bits [9:5]

The source register value from the issued instruction, containing either the source address or the source data.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

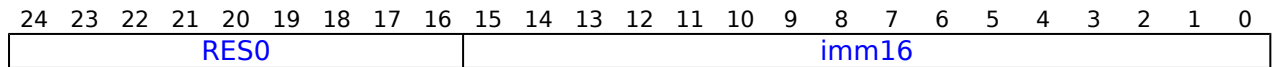
sizereg, bits [4:0]

The size register value from the issued instruction, containing the number of bytes to be transferred or set.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from HVC or SVC instruction execution



Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the HVC or SVC instruction.

For an HVC instruction, and for an A64 SVC instruction, this is the value of the imm16 field of the issued instruction.

For an A32 or T32 SVC instruction:

- If the instruction is unconditional, then:
 - For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
 - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- If the instruction is conditional, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

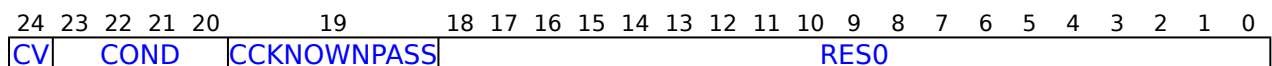
In AArch32 state, the HVC instruction is unconditional, and a conditional SVC instruction generates an exception only if it passes its condition code check. Therefore, the syndrome information for these exceptions does not require conditionality information.

For T32 and A32 instructions, see 'SVC' and 'HVC'.

For A64 instructions, see 'SVC' and 'HVC'.

If FEAT_FGT is implemented, [HFGITR_EL2](#).{SVC_EL1, SVC_EL0} control fine-grained traps on SVC execution.

ISS encoding for an exception from SMC instruction execution in AArch32 state



For an SMC instruction that completes normally and generates an exception that is taken to EL3, the ISS encoding is RES0.

For an SMC instruction that is trapped to EL2 from EL1 because [HCR_EL2](#).TSC is 1, the ISS encoding is as shown in the diagram.

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CCKNOWNPASS, bit [19]

Indicates whether the instruction might have failed its condition code check.

| CCKNOWNPASS | Meaning |
|-------------|--|
| 0b0 | The instruction was unconditional, or was conditional and passed its condition code check. |
| 0b1 | The instruction was conditional, and might have failed its condition code check. |

Note

In an implementation in which an SMC instruction that fails its code check is not trapped, this field can always return the value 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [18:0]

Reserved, RES0.

[HCR_EL2](#).TSC describes the configuration settings for trapping SMC instructions to EL2.

'System calls' describes the case where these exceptions are trapped to EL3.

ISS encoding for an exception from SMC instruction execution in AArch64 state

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | imm16 | | | | | | | | | | | | | | |

Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the issued SMC instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The value of ISS[24:0] described here is used both:

- When an SMC instruction is trapped from EL1 modes.
- When an SMC instruction is not trapped, so completes normally and generates an exception that is taken to EL3.

[HCR_EL2](#).TSC describes the configuration settings for trapping SMC from EL1 modes.

'System calls' describes the case where these exceptions are trapped to EL3.

ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|-----|----|-----|----|-----|----|-----|----|----|----|----|----|-----|---|---|-----------|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | Op0 | | Op2 | | Op1 | | CRn | | | Rt | | | CRm | | | Direction | | | | | | |

Bits [24:22]

Reserved, RES0.

Op0, bits [21:20]

The Op0 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op2, bits [19:17]

The Op2 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op1, bits [16:14]

The Op1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|---|
| 0b0 | Write access, including MSR instructions. |
| 0b1 | Read access, including MRS instructions. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For exceptions caused by System instructions, see 'System instructions' subsection of 'Branches, exception generating and System instructions' for the encoding values returned by an instruction.

The following fields describe configuration settings for generating the exception that is reported using EC value 0b011000:

- [SCTLR_EL1](#).UCI, for execution of cache maintenance instructions using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [SCTLR_EL1](#).UCT, for accesses to [CTR_EL0](#) using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [SCTLR_EL1](#).DZE, for execution of DC ZVA instructions using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [SCTLR_EL1](#).UMA, for accesses to the PSTATE interrupt masks using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [CPACR_EL1](#).TTA, for accesses to the trace registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [MDSCR_EL1](#).TDCC, for accesses to the Debug Communications Channel (DCC) registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.

- If FEAT_FGT is implemented, [MDCR_EL2](#).TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and [MDCR_EL3](#).TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.
- [CNTCTL_EL1](#).{ELOPTEN, EL0VTEN, EL0PCTEN, EL0VCTEN} accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [PMUSERENR_EL0](#).{ER, CR, SW, EN}, for accesses to the Performance Monitor registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [AMUSERENR_EL0](#).EN, for accesses to Activity Monitors registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [HCR_EL2](#).{TRVM, TVM}, for accesses to virtual memory control registers using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).TDZ, for execution of DC ZVA instructions using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).TTLB, for execution of TLB maintenance instructions using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).{TSW, TPC, TPU}, for execution of cache maintenance instructions using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).TACR, for accesses to the Auxiliary Control Register, [ACTLR_EL1](#), using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).TIDCP, for accesses to lockdown, DMA, and TCM operations using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).{TID1, TID2, TID3}, for accesses to ID group 1, ID group 2 or ID group 3 registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [CPTR_EL2](#).TCPAC, for accesses to [CPACR_EL1](#), using AArch64 state, MSR or MRS access trapped to EL2.
- [CPTR_EL2](#).TTA, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).TTRF, for accesses to the trace filter control register, [TRFCR_EL1](#), using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).TDRA, for accesses to Debug ROM registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).TDOSA, for accesses to powerdown debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- [CNTHCTL_EL2](#).{EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).TDA, for accesses to debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).{TPM, TPMCR}, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [CPTR_EL2](#).TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).APK, for accesses to Pointer authentication key registers. using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).{NV, NV1}, for Nested virtualization register access, using AArch64 state, MSR or MRS access, trapped to EL2.
- [HCR_EL2](#).AT, for execution of AT S1E* instructions, using AArch64 state, MSR or MRS access, trapped to EL2.
- [HCR_EL2](#).{TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access, trapped to EL2.
- [SCR_EL3](#).APK, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [SCR_EL3](#).ST, for accesses to the Counter-timer Physical Secure timer registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [SCR_EL3](#).{TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [CPTR_EL3](#).TCPAC, for accesses to [CPTR_EL2](#) and [CPACR_EL1](#) using AArch64 state, MSR or MRS access trapped to EL3.
- [CPTR_EL3](#).TTA, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [MDCR_EL3](#).TTRF, for accesses to the trace filter control registers, [TRFCR_EL1](#) and [TRFCR_EL2](#), using AArch64 state, MSR or MRS access trapped to EL3.
- [MDCR_EL3](#).TDA, for accesses to debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [MDCR_EL3](#).TDOSA, for accesses to powerdown debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [MDCR_EL3](#).TPM, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [CPTR_EL3](#).TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access, trapped to EL3.

- If FEAT_EVT is implemented, the following registers control traps for EL1 and EL0 Cache controls that use this EC value:
 - [HCR_EL2](#).{TTLBOS, TTLBIS, TICAB, TOCU, TID4}.
 - [HCR2](#).{TTLBIS, TICAB, TOCU, TID4}.
- If FEAT_FGT is implemented:
 - [SCR_EL3](#).FGTEn, for accesses to the fine-grained trap registers, MSR or MRS access at EL2 trapped to EL3.
 - [HFGTR_EL2](#) for reads and [HFGWTR_EL2](#) for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 trapped to EL2.
 - [HFGITR_EL2](#) for execution of system instructions, MSR or MRS access trapped to EL2
 - [HDFGTR_EL2](#) for reads and [HDFGWTR_EL2](#) for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 state trapped to EL2.
 - [HAFGTR_EL2](#) for reads of Activity Monitor counters, using AArch64 state, MRS access at EL0 and EL1 trapped to EL2.
- If FEAT_RNG_TRAP is implemented:
 - [SCR_EL3](#).TRNDR for reads of [RNDR](#) and [RNDRRS](#) using AArch64 state, MRS access trapped to EL3.
- If FEAT_SME is implemented:
 - [CPTR_EL3](#).ESM, for MSR or MRS accesses to [SMPRI_EL1](#) at EL1, EL2, and EL3, trapped to EL3.
 - [CPTR_EL3](#).ESM, for MSR or MRS accesses to [SMPRMAP_EL2](#) at EL2 and EL3, trapped to EL3.
 - [SCTLR_EL1](#).EnTP2, for MSR or MRS accesses to [TPIDR2_EL0](#) at EL0, trapped to EL1 or EL2.
 - [SCTLR_EL2](#).EnTP2, for MSR or MRS accesses to [TPIDR2_EL0](#) at EL0, trapped to EL2.
 - [SCR_EL3](#).EnTP2, for MSR or MRS accesses to [TPIDR2_EL0](#) at EL0, EL1, and EL2, trapped to EL3.
- If FEAT_NMI is implemented, [HCRX_EL2](#).TALLINT, for MSR writes of [ALLINT](#) at EL1, trapped to EL2.

ISS encoding for an IMPLEMENTATION DEFINED exception to EL3

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [24:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from an Instruction Abort

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|-----|-----|----|------|-------|------|------|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | SET | FnV | EA | RES0 | S1PTW | RES0 | IFSC | | | | | | |

Bits [24:13]

Reserved, RES0.

SET, bits [12:11]

When FEAT_RAS is implemented:

Synchronous Error Type. When IFSC is 0b010000, describes the PE error state after taking the Instruction Abort exception.

| SET | Meaning |
|------|--------------------------|
| 0b00 | Recoverable state (UER). |
| 0b10 | Uncontainable (UC). |
| 0b11 | Restartable state (UEO). |

All other values are reserved.

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in a PE state that is not recoverable.

This field is valid only if the IFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|---|
| 0b0 | FAR is valid. |
| 0b1 | FAR is not valid, and holds an UNKNOWN value. |

This field is valid only if the IFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

| S1PTW | Meaning |
|-------|---|
| 0b0 | Fault not on a stage 2 translation for a stage 1 translation table walk. |
| 0b1 | Fault on the stage 2 translation of an access for a stage 1 translation table walk. |

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code.

| IFSC | Meaning | Applies when |
|----------|---|--|
| 0b000000 | Address size fault, level 0 of translation or translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000100 | Translation fault, level 0. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001000 | Access flag fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001100 | Permission fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010000 | Synchronous External abort, not on translation table walk or hardware update of translation table. | |
| 0b010011 | Synchronous External abort on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented |
| 0b010100 | Synchronous External abort on translation table walk or hardware update of translation table, level 0. | |
| 0b010101 | Synchronous External abort on translation table walk or hardware update of translation table, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk or hardware update of translation table, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk or hardware update of translation table, level 3. | |
| 0b011000 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b011011 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented and FEAT_RAS is not implemented |
| 0b011100 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0. | When FEAT_RAS is not implemented |
| 0b011101 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1. | When FEAT_RAS is not implemented |
| 0b011110 | Synchronous parity or ECC error on memory access on translation table walk or | When FEAT_RAS is |

| | | |
|----------|---|---|
| 0b011111 | hardware update of translation table, level 2. Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3. | not implemented When FEAT_RAS is not implemented |
| 0b100011 | Granule Protection Fault on translation table walk or hardware update of translation table, level -1. | When FEAT_RME is implemented and FEAT_LPA2 is implemented |
| 0b100100 | Granule Protection Fault on translation table walk or hardware update of translation table, level 0. | When FEAT_RME is implemented |
| 0b100101 | Granule Protection Fault on translation table walk or hardware update of translation table, level 1. | When FEAT_RME is implemented |
| 0b100110 | Granule Protection Fault on translation table walk or hardware update of translation table, level 2. | When FEAT_RME is implemented |
| 0b100111 | Granule Protection Fault on translation table walk or hardware update of translation table, level 3. | When FEAT_RME is implemented |
| 0b101000 | Granule Protection Fault, not on translation table walk or hardware update of translation table. | When FEAT_RME is implemented |
| 0b101001 | Address size fault, level -1. | When FEAT_LPA2 is implemented |
| 0b101011 | Translation fault, level -1. | When FEAT_LPA2 is implemented |
| 0b110000 | TLB conflict abort. | |
| 0b110001 | Unsupported atomic hardware update fault. | When FEAT_HAFDBS is implemented |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults'.

Note

Because Access flag faults and Permission faults can result only from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception due to SME functionality

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|------|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | SMTc | | |

The accesses covered by this trap include:

- Execution of SME instructions.
- Execution of SVE and Advanced SIMD instructions, when the PE is in Streaming SVE mode.
- Direct accesses of [SVCR](#), [SMCR_EL1](#), [SMCR_EL2](#), [SMCR_EL3](#).

Bits [24:3]

Reserved, RES0.

SMTC, bits [2:0]

SME Trap Code. Identifies the reason for instruction trapping.

| SMTC | Meaning |
|-------|--|
| 0b000 | Access to SME functionality trapped as a result of CPACR_EL1 .SMEN, CPTR_EL2 .SMEN, CPTR_EL2 .TSM, or CPTR_EL3 .ESM, that is not reported using EC 0b000000. |
| 0b001 | Advanced SIMD, SVE, or SVE2 instruction trapped because PSTATE.SM is 1. |
| 0b010 | SME instruction trapped because PSTATE.SM is 0. |
| 0b011 | SME instruction trapped because PSTATE.ZA is 0. |

All other values are reserved.

The following fields describe the configuration settings for the traps that are reported using the EC value 0b011101:

- [CPACR_EL1](#).SMEN, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access [SVCR](#) and [SMCR_EL1](#) System registers at EL1 and EL0, trapped to EL1 or EL2.
- [CPTR_EL2](#).SMEN and [CPTR_EL2](#).TSM, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access [SVCR](#), [SMCR_EL1](#), [SMCR_EL2](#) at EL2, EL1, or EL0, trapped to EL2.
- [CPTR_EL3](#).ESM, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access [SVCR](#), [SMCR_EL1](#), [SMCR_EL2](#), [SMCR_EL3](#) from all Exception levels and any Security state, trapped to EL3.

ISS encoding for an exception from a Granule Protection Check

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|-------|----|-----|-------|----|----|------|----|------|------|----|-------|-----|------|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | S2PTW | | InD | GPCSC | | | VNCR | | RES0 | RES0 | CM | S1PTW | WnR | xFSC | | | | | | | | |

Bits [24:22]

Reserved, RES0.

S2PTW, bit [21]

Indicates whether the Granule Protection Check exception was on an access made for a stage 2 translation table walk.

| S2PTW | Meaning |
|-------|--|
| 0b0 | Fault not on a stage 2 translation table walk. |
| 0b1 | Fault on a stage 2 translation table walk. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

InD, bit [20]

Indicates whether the Granule Protection Check exception was on an instruction or data access.

| InD | Meaning |
|-----|---------------------|
| 0b0 | Data access. |
| 0b1 | Instruction access. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GPCSC, bits [19:14]

Granule Protection Check Status Code.

| GPCSC | Meaning |
|----------|---|
| 0b000000 | GPT address size fault at level 0. |
| 0b000001 | GPT address size fault at level 1. |
| 0b000100 | GPT walk fault at level 0. |
| 0b000101 | GPT walk fault at level 1. |
| 0b001100 | Granule protection fault at level 0. |
| 0b001101 | Granule protection fault at level 1. |
| 0b010100 | Synchronous External abort on GPT fetch at level 0. |
| 0b010101 | Synchronous External abort on GPT fetch at level 1. |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VNCR, bit [13]

When FEAT_NV2 is implemented:

Indicates that the fault came from use of [VNCR_EL2](#) register by EL1 code.

| VNCR | Meaning |
|------|--|
| 0b0 | The fault was not generated by the use of VNCR_EL2 , by an MRS or MSR instruction executed at EL1. |
| 0b1 | The fault was generated by the use of VNCR_EL2 , by an MRS or MSR instruction executed at EL1. |

This field is 0 in ESR_EL1.

When InD is '1', this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [12:11]

Reserved, RES0.

Bits [10:9]

Reserved, RES0.

CM, bit [8]

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:

| CM | Meaning |
|-----------|--|
| 0b0 | The Data Abort was not generated by the execution of one of the System instructions identified in the description of value 1. |
| 0b1 | The Data Abort was generated by either the execution of a cache maintenance instruction or by a synchronous fault on the execution of an address translation instruction. The DC ZVA , DC GVA , and DC GZVA instructions are not classified as cache maintenance instructions, and therefore their execution cannot cause this field to be set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

Indicates whether the Granule Protection Check exception was on an access for stage 2 translation for a stage 1 translation table walk:

| S1PTW | Meaning |
|--------------|---|
| 0b0 | Fault not on a stage 2 translation for a stage 1 translation table walk. |
| 0b1 | Fault on the stage 2 translation of an access for a stage 1 translation table walk. |

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

| WnR | Meaning |
|------------|--|
| 0b0 | Abort caused by an instruction reading from a memory location. |
| 0b1 | Abort caused by an instruction writing to a memory location. |

When InD is '1', this field is RES0.

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is UNKNOWN for:

- An External abort on an Atomic access.
- A fault reported using a DFSC value of 0b110101 or 0b110001, indicating an unsupported Exclusive or atomic access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

xFSC, bits [5:0]

Instruction or Data Fault Status Code.

| xFSC | Meaning | Applies when |
|-------------|---|---|
| 0b100011 | Granule Protection Fault on translation table walk or hardware update of translation table, level -1. | When FEAT_RME is implemented and FEAT_LPA2 is implemented |
| 0b100100 | Granule Protection Fault on translation table walk or hardware update of translation table, level 0. | When FEAT_RME is implemented |
| 0b100101 | Granule Protection Fault on translation table walk or hardware update of translation table, level 1. | When FEAT_RME is implemented |
| 0b100110 | Granule Protection Fault on translation table walk or hardware update of translation table, level 2. | When FEAT_RME is implemented |
| 0b100111 | Granule Protection Fault on translation table walk or hardware update of translation table, level 3. | When FEAT_RME is implemented |
| 0b101000 | Granule Protection Fault, not on translation table walk or hardware update of translation table. | When FEAT_RME is implemented |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults'.

Note

Because Access flag faults and Permission faults can result only from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from a Data Abort

| | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|-----|-----|-----|----|----|----|----|------|-------------|----|----|-----|----|----|-------|-----|------|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ISV | SAS | SSE | SRT | | | SF | AR | VNCR | Bits[12:11] | | | FnV | EA | CM | S1PTW | WnR | DFSC | | | | | | | |

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV or ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this ISS encoding includes ISS2, bits[36:32].

ISV, bit [24]

Instruction Syndrome Valid. Indicates whether the syndrome information in ISS[23:14] is valid.

| ISV | Meaning |
|-----|---|
| 0b0 | No valid instruction syndrome. ISS[23:14] are RES0. |
| 0b1 | ISS[23:14] hold a valid instruction syndrome. |

In ESR_EL2, ISV is 1 when FEAT_LS64 is implemented and a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

For other faults reported in ESR_EL2, ISV is 0 except for the following stage 2 aborts:

- AArch64 loads and stores of a single general-purpose register (including the register specified with 0b11111, including those with Acquire/Release semantics, but excluding Load Exclusive or Store Exclusive and excluding those with writeback).
- AArch32 instructions where the instruction:
 - Is an LDR, LDA, LDRT, LDRSH, LDRSHT, LDRH, LDAH, LDRHT, LDRSB, LDRSBT, LDRB, LDAB, LDRBT, STR, STL, STRT, STRH, STLH, STRHT, STRB, STLB, or STRBT instruction.
 - Is not performing register writeback.
 - Is not using R15 as a source or destination register.

For these stage 2 aborts, ISV is UNKNOWN if the exception was generated in Debug state in memory access mode, and otherwise indicates whether ISS[23:14] hold a valid syndrome.

For faults reported in ESR_EL1 or ESR_EL3, ISV is 1 when FEAT_LS64 is implemented and a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault. ISV is 0 for all other faults reported in ESR_EL1 or ESR_EL3.

When FEAT_RAS is implemented, ISV is 0 for any synchronous External abort.

For ISS reporting, a stage 2 abort on a stage 1 translation table walk does not return a valid instruction syndrome, and therefore ISV is 0 for these aborts.

When FEAT_RAS is not implemented, it is IMPLEMENTATION DEFINED whether ISV is set to 1 or 0 on a synchronous External abort on a stage 2 translation table walk.

When FEAT_MTE2 is implemented, for a synchronous Tag Check Fault abort taken to ELx, ESR_ELx.FNV is 0 and FAR_ELx is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SAS, bits [23:22]

When ISV == 1:

Syndrome Access Size. Indicates the size of the access attempted by the faulting operation.

| SAS | Meaning |
|------|------------|
| 0b00 | Byte |
| 0b01 | Halfword |
| 0b10 | Word |
| 0b11 | Doubleword |

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0b11.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSE, bit [21]
When ISV == 1:

Syndrome Sign Extend. For a byte, halfword, or word load operation, indicates whether the data item must be sign extended.

| SSE | Meaning |
|-----|----------------------------------|
| 0b0 | Sign-extension not required. |
| 0b1 | Data item must be sign-extended. |

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

For all other operations, this field is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SRT, bits [20:16]
When ISV == 1:

Syndrome Register Transfer. The register number of the Wt/Xt/Rt operand of the faulting instruction.

If the exception was taken from an Exception level that is using AArch32, then this is the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states'.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SF, bit [15]
When ISV == 1:

Width of the register accessed by the instruction is Sixty-Four.

| SF | Meaning |
|-----|--|
| 0b0 | Instruction loads/stores a 32-bit wide register. |
| 0b1 | Instruction loads/stores a 64-bit wide register. |

Note

This field specifies the register width identified by the instruction, not the Execution state.

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 1.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AR, bit [14]

When ISV == 1:

Acquire/Release.

| AR | Meaning |
|-----|---|
| 0b0 | Instruction did not have acquire/release semantics. |
| 0b1 | Instruction did have acquire/release semantics. |

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VNCR, bit [13]

When FEAT_NV2 is implemented:

Indicates that the fault came from use of [VNCR_EL2](#) register by EL1 code.

| VNCR | Meaning |
|------|--|
| 0b0 | The fault was not generated by the use of VNCR_EL2 , by an MRS or MSR instruction executed at EL1. |
| 0b1 | The fault was generated by the use of VNCR_EL2 , by an MRS or MSR instruction executed at EL1. |

This field is 0 in ESR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SET, bits [12:11]

When FEAT_RAS is implemented and FEAT_LS64 is not implemented:

Synchronous Error Type. When DFSC is 0b010000, describes the PE error state after taking the Data Abort exception.

| SET | Meaning |
|------|--------------------------|
| 0b00 | Recoverable state (UER). |
| 0b10 | Uncontainable (UC). |
| 0b11 | Restartable state (UEO). |

All other values are reserved.

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in a PE state that is not recoverable.

This field is valid only if the DFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_LS64 is implemented:

Load/Store Type. Used when an LD64B, ST64B, ST64BV, or ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

| LST | Meaning |
|------|---|
| 0b01 | An ST64BV instruction generated the Data Abort. |
| 0b10 | An LD64B or ST64B instruction generated the Data Abort. |
| 0b11 | An ST64BV0 instruction generated the Data Abort. |

All other values are reserved.

This field is valid only if the DFSC code is 0b110101. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|---|
| 0b0 | FAR is valid. |
| 0b1 | FAR is not valid, and holds an UNKNOWN value. |

This field is valid only if the DFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CM, bit [8]

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:

| CM | Meaning |
|-----|--|
| 0b0 | The Data Abort was not generated by the execution of one of the System instructions identified in the description of value 1. |
| 0b1 | The Data Abort was generated by either the execution of a cache maintenance instruction or by a synchronous fault on the execution of an address translation instruction. The DC ZVA , DC GVA , and DC GZVA instructions are not classified as cache maintenance instructions, and therefore their execution cannot cause this field to be set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

| S1PTW | Meaning |
|-------|---|
| 0b0 | Fault not on a stage 2 translation for a stage 1 translation table walk. |
| 0b1 | Fault on the stage 2 translation of an access for a stage 1 translation table walk. |

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

| WnR | Meaning |
|-----|--|
| 0b0 | Abort caused by an instruction reading from a memory location. |
| 0b1 | Abort caused by an instruction writing to a memory location. |

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is UNKNOWN for:

- An External abort on an Atomic access.
- A fault reported using a DFSC value of 0b110101 or 0b110001, indicating an unsupported Exclusive or atomic access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code.

| DFSC | Meaning | Applies when |
|----------|---|--|
| 0b000000 | Address size fault, level 0 of translation or translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000100 | Translation fault, level 0. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001000 | Access flag fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001100 | Permission fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010000 | Synchronous External abort, not on translation table walk or hardware update of translation table. | |
| 0b010001 | Synchronous Tag Check Fault. | When FEAT_MTE2 is implemented |
| 0b010011 | Synchronous External abort on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented |
| 0b010100 | Synchronous External abort on translation table walk or hardware update of translation table, level 0. | |
| 0b010101 | Synchronous External abort on translation table walk or hardware update of translation table, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk or hardware update of translation table, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk or hardware update of translation table, level 3. | |
| 0b011000 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b011011 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented and FEAT_RAS is not implemented |
| 0b011100 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0. | When FEAT_RAS is not implemented |
| 0b011101 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1. | When FEAT_RAS is not implemented |

| | | |
|-----------|--|---|
| 0b0111110 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2. | When FEAT_RAS is not implemented |
| 0b0111111 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3. | When FEAT_RAS is not implemented |
| 0b100001 | Alignment fault. | |
| 0b100011 | Granule Protection Fault on translation table walk or hardware update of translation table, level -1. | When FEAT_RME is implemented and FEAT_LPA2 is implemented |
| 0b100100 | Granule Protection Fault on translation table walk or hardware update of translation table, level 0. | When FEAT_RME is implemented |
| 0b100101 | Granule Protection Fault on translation table walk or hardware update of translation table, level 1. | When FEAT_RME is implemented |
| 0b100110 | Granule Protection Fault on translation table walk or hardware update of translation table, level 2. | When FEAT_RME is implemented |
| 0b100111 | Granule Protection Fault on translation table walk or hardware update of translation table, level 3. | When FEAT_RME is implemented |
| 0b101000 | Granule Protection Fault, not on translation table walk or hardware update of translation table. | When FEAT_RME is implemented |
| 0b101001 | Address size fault, level -1. | When FEAT_LPA2 is implemented |
| 0b101011 | Translation fault, level -1. | When FEAT_LPA2 is implemented |
| 0b110000 | TLB conflict abort. | |
| 0b110001 | Unsupported atomic hardware update fault. | When FEAT_HAFDBS is implemented |
| 0b110100 | IMPLEMENTATION DEFINED fault (Lockdown). | |
| 0b110101 | IMPLEMENTATION DEFINED fault (Unsupported Exclusive or Atomic access). | |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults'.

Note

Because Access flag faults and Permission faults can result only from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from a trapped floating-point exception

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|----|----|----|----|----|----|------|----|----|----|----|----|--------|-----|------|-----|-----|-----|-----|-----|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | TFV | | | | | | | RES0 | | | | | | VECITR | IDF | RES0 | IXF | UFF | OFF | DZF | IOF | | | |

Bit [24]

Reserved, RES0.

TFV, bit [23]

Trapped Fault Valid bit. Indicates whether the IDF, IXF, UFF, OFF, DZF, and IOF bits hold valid information about trapped floating-point exceptions.

| TFV | Meaning |
|-----|---|
| 0b0 | The IDF, IXF, UFF, OFF, DZF, and IOF bits do not hold valid information about trapped floating-point exceptions and are UNKNOWN. |
| 0b1 | One or more floating-point exceptions occurred during an operation performed while executing the reported instruction. The IDF, IXF, UFF, OFF, DZF, and IOF bits indicate trapped floating-point exceptions that occurred. For more information, see 'Floating-point exceptions and exception traps'. |

It is IMPLEMENTATION DEFINED whether this field is set to 0 on an exception generated by a trapped floating-point exception from an instruction that is performing floating-point operations on more than one lane of a vector.

Note

This is not a requirement. Implementations can set this field to 1 on a trapped floating-point exception from an instruction and return valid information in the {IDF, IXF, UFF, OFF, DZF, IOF} fields.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [22:11]

Reserved, RES0.

VECITR, bits [10:8]

For a trapped floating-point exception from an instruction executed in AArch32 state this field is RES1.

For a trapped floating-point exception from an instruction executed in AArch64 state this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| IDF | Meaning |
|------------|--|
| 0b0 | Input denormal floating-point exception has not occurred. |
| 0b1 | Input denormal floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| IXF | Meaning |
|------------|---|
| 0b0 | Inexact floating-point exception has not occurred. |
| 0b1 | Inexact floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFF, bit [3]

Underflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| UFF | Meaning |
|------------|---|
| 0b0 | Underflow floating-point exception has not occurred. |
| 0b1 | Underflow floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFF, bit [2]

Overflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| OFF | Meaning |
|------------|--|
| 0b0 | Overflow floating-point exception has not occurred. |
| 0b1 | Overflow floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZF, bit [1]

Divide by Zero floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| DZF | Meaning |
|-----|--|
| 0b0 | Divide by Zero floating-point exception has not occurred. |
| 0b1 | Divide by Zero floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOF, bit [0]

Invalid Operation floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| IOF | Meaning |
|-----|---|
| 0b0 | Invalid Operation floating-point exception has not occurred. |
| 0b1 | Invalid Operation floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

In an implementation that supports the trapping of floating-point exceptions:

- From an Exception level using AArch64, the [FPCR](#).{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.
- From an Exception level using AArch32, the [FPSCR](#).{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.

ISS encoding for an SError interrupt

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|------|----|----|----|----|----|----|----|----|----|------|----|-----|---|----|---|------|---|---|------|---|---|---|--|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| IDS | | RES0 | | | | | | | | | | IESB | | AET | | EA | | RES0 | | | DFSC | | | | |

IDS, bit [24]

IMPLEMENTATION DEFINED syndrome.

| IDS | Meaning |
|---|---|
| 0b0 | Bits [23:0] of the ISS field holds the fields described in this encoding. |
| Note If FEAT_RAS is not implemented, bits [23:0] of the ISS field are RES0. | |
| 0b1 | Bits [23:0] of the ISS field holds IMPLEMENTATION DEFINED syndrome information that can be used to provide additional information about the SError interrupt. |

Note

This field was previously called ISV.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:14]

Reserved, RES0.

IESB, bit [13]

When **FEAT_IESB** is implemented:

Implicit error synchronization event.

| IESB | Meaning |
|-------------|--|
| 0b0 | The SError interrupt was either not synchronized by the implicit error synchronization event or not taken immediately. |
| 0b1 | The SError interrupt was synchronized by the implicit error synchronization event and taken immediately. |

This field is valid only if the DFSC code is 0b010001. It is RES0 for all other errors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AET, bits [12:10]

When **FEAT_RAS** is implemented:

Asynchronous Error Type.

When DFSC is 0b010001, describes the PE error state after taking the SError interrupt exception.

| AET | Meaning |
|------------|----------------------------|
| 0b000 | Uncontainable (UC). |
| 0b001 | Unrecoverable state (UEU). |
| 0b010 | Restartable state (UEO). |
| 0b011 | Recoverable state (UER). |
| 0b110 | Corrected (CE). |

All other values are reserved.

If multiple errors are taken as a single SError interrupt exception, the overall PE error state is reported.

Note

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

This field is valid only if the DFSC code is 0b010001. It is RES0 for all other errors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EA, bit [9]**When FEAT_RAS is implemented:**

External abort type. When DFSC is 0b010001, provides an IMPLEMENTATION DEFINED classification of External aborts.

This field is valid only if the DFSC code is 0b010001. It is RES0 for all other errors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [8:6]

Reserved, RES0.

DFSC, bits [5:0]**When FEAT_RAS is implemented:**

Data Fault Status Code.

| DFSC | Meaning |
|----------|--------------------------------|
| 0b000000 | Uncategorized error. |
| 0b010001 | Asynchronous SError interrupt. |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ISS encoding for an exception from a Breakpoint or Vector Catch debug exception

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|------|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | IFSC | | | | | |

Bits [24:6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code.

| IFSC | Meaning |
|----------|------------------|
| 0b100010 | Debug exception. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions:

- For exceptions from AArch64, see 'Breakpoint exceptions'.
- For exceptions from AArch32, see 'Breakpoint exceptions' and 'Vector Catch exceptions'.

ISS encoding for an exception from a Software Step exception

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|----|---|------|---|---|---|--|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| ISV | | RES0 | | | | | | | | | | | | | | | | | EX | | IFSC | | | | |

ISV, bit [24]

Instruction syndrome valid. Indicates whether the EX bit, ISS[6], is valid, as follows:

| ISV | Meaning |
|-----|------------------|
| 0b0 | EX bit is RES0. |
| 0b1 | EX bit is valid. |

See the EX bit description for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:7]

Reserved, RES0.

EX, bit [6]

Exclusive operation. If the ISV bit is set to 1, this bit indicates whether a Load-Exclusive instruction was stepped.

| EX | Meaning |
|-----|---|
| 0b0 | An instruction other than a Load-Exclusive instruction was stepped. |
| 0b1 | A Load-Exclusive instruction was stepped. |

If the ISV bit is set to 0, this bit is RES0, indicating no syndrome data is available.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IFSC, bits [5:0]

Instruction Fault Status Code.

| IFSC | Meaning |
|----------|------------------|
| 0b100010 | Debug exception. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Software Step exceptions'.

ISS encoding for an exception from a Watchpoint exception

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|------|------|------|----|----|---|---|----|------|-----|------|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | RES0 | VNCR | RES0 | | | | | CM | RES0 | WnR | DFSC | | | | |

Bits [24:15]

Reserved, RES0.

Bit [14]

Reserved, RES0.

VNCR, bit [13]

When FEAT_NV2 is implemented:

Indicates that the watchpoint came from use of [VNCR_EL2](#) register by EL1 code.

| VNCR | Meaning |
|------|--|
| 0b0 | The watchpoint was not generated by the use of VNCR_EL2 by EL1 code. |
| 0b1 | The watchpoint was generated by the use of VNCR_EL2 by EL1 code. |

This field is 0 in ESR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [12:9]

Reserved, RES0.

CM, bit [8]

Cache maintenance. Indicates whether the Watchpoint exception came from a cache maintenance or address translation instruction:

| CM | Meaning |
|-----|---|
| 0b0 | The Watchpoint exception was not generated by the execution of one of the System instructions identified in the description of value 1. |
| 0b1 | The Watchpoint exception was generated by either the execution of a cache maintenance instruction or by a synchronous Watchpoint exception on the execution of an address translation instruction. The DC ZVA , DC GVA , and DC GZVA instructions are not classified as a cache maintenance instructions, and therefore their execution cannot cause this field to be set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

WnR, bit [6]

Write not Read. Indicates whether the Watchpoint exception was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

| WnR | Meaning |
|-----|---|
| 0b0 | Watchpoint exception caused by an instruction reading from a memory location. |
| 0b1 | Watchpoint exception caused by an instruction writing to a memory location. |

For Watchpoint exceptions on cache maintenance and address translation instructions, this bit always returns a value of 1.

For Watchpoint exceptions from an atomic instruction, this field is set to 0 if a read of the location would have generated the Watchpoint exception, otherwise it is set to 1.

If multiple watchpoints match on the same access, it is UNPREDICTABLE which watchpoint generates the Watchpoint exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code.

| DFSC | Meaning |
|----------|------------------|
| 0b100010 | Debug exception. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Watchpoint exceptions'.

ISS encoding for an exception from execution of a Breakpoint instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | Comment | | | | | | | | | | | | | | |

Bits [24:16]

Reserved, RES0.

Comment, bits [15:0]

Set to the instruction comment field value, zero extended as necessary.

For the AArch32 BKPT instructions, the comment field is described as the immediate field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Breakpoint instruction exceptions'.

ISS encoding for an exception from an ERET, ERETAA, or ERETAB instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|------|---|-------|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | ERET | | ERETA |

This EC value applies when FEAT_FGT is implemented, or when [HCR_EL2.NV](#) is 1.

Bits [24:2]

Reserved, RES0.

ERET, bit [1]

Indicates whether an ERET or ERETA* instruction was trapped to EL2.

| ERET | Meaning |
|------|--|
| 0b0 | ERET instruction trapped to EL2. |
| 0b1 | ERETAA or ERETAB instruction trapped to EL2. |

If this bit is 0, the ERETA field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ERETA, bit [0]

Indicates whether an ERETAA or ERETAB instruction was trapped to EL2.

| ERETA | Meaning |
|-------|------------------------------------|
| 0b0 | ERETAA instruction trapped to EL2. |
| 0b1 | ERETAB instruction trapped to EL2. |

When the ERET field is 0, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see [HCR_EL2.NV](#).

If FEAT_FGT is implemented, [HFGITR_EL2.ERET](#) controls fine-grained trap exceptions from ERET, ERETAA and ERETAB execution.

ISS encoding for an exception from a TSTART instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|------|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | Rd | | | RES0 | | | | | | |

Bits [24:10]

Reserved, RES0.

Rd, bits [9:5]

The Rd value from the issued instruction, the general purpose register used for the destination.

Bits [4:0]

Reserved, RES0.

ISS encoding for an exception from Branch Target Identification instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|------|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | BTYP | | |

Bits [24:2]

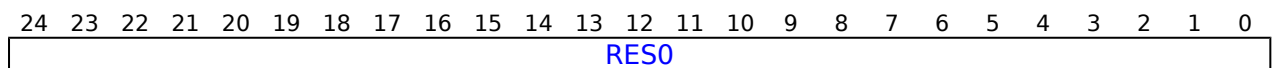
Reserved, RES0.

BTYPE, bits [1:0]

This field is set to the PSTATE.BTYPE value that generated the Branch Target Exception.

For more information about generating these exceptions, see 'The AArch64 application level programmers model'.

ISS encoding for an exception from a Pointer Authentication instruction when HCR_EL2.API == 0 || SCR_EL3.API == 0

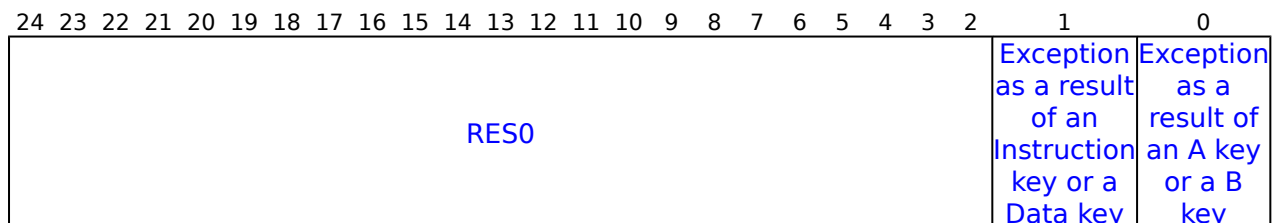
**Bits [24:0]**

Reserved, RES0.

For more information about generating these exceptions, see:

- [HCR_EL2.API](#), for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL2.
- [SCR_EL3.API](#), for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL3.

ISS encoding for an exception from a Pointer Authentication instruction authentication failure

**Bits [24:2]**

Reserved, RES0.

Bit [1]

This field indicates whether the exception is as a result of an Instruction key or a Data key.

| Meaning | |
|---------|------------------|
| 0b0 | Instruction Key. |
| 0b1 | Data Key. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

This field indicates whether the exception is as a result of an A key or a B key.

| | Meaning |
|-----|---------|
| 0b0 | A key. |
| 0b1 | B key. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following instructions generate an exception when the Pointer Authentication Code (PAC) is incorrect:

- AUTIASP, AUTIAZ, AUTIA1716.
- AUTIBSP, AUTIBZ, AUTIB1716.
- AUTIA, AUTDA, AUTIB, AUTDB.
- AUTIZA, AUTIZB, AUTDZA, AUTDZB.

It is IMPLEMENTATION DEFINED whether the following instructions generate an exception directly from the authorization failure, rather than changing the address in a way that will generate a Translation fault when the address is accessed:

- RETAA, RETAB.
- BRAA, BRAB, BLRAA, BLRAB.
- BRAAZ, BRABZ, BLRAAZ, BLRABZ.
- ERETAA, ERETAB.
- LDRAA, LDRAB, whether the authenticated address is written back to the base register or not.

Accessing ESR_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic ESR_EL1 or ESR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ESR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ESR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x138];
    else
        return ESR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return ESR_EL2;
    else
        return ESR_EL1;
elsif PSTATE.EL == EL3 then
    return ESR_EL1;

```

MSR ESR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ESR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x138] = X[t];
    else
        ESR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' then
            ESR_EL2 = X[t];
        else
            ESR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        ESR_EL1 = X[t];

```

MRS <Xt>, ESR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x138];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' then
            return ESR_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
            return ESR_EL1;
        else
            UNDEFINED;

```

MSR ESR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x138] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        ESR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        ESR_EL1 = X[t];
    else
        UNDEFINED;

```

MRS <Xt>, ESR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return ESR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return ESR_EL2;
elsif PSTATE.EL == EL3 then
    return ESR_EL2;

```

MSR ESR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        ESR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ESR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    ESR_EL2 = X[t];

```


ESR_EL2, Exception Syndrome Register (EL2)

The ESR_EL2 characteristics are:

Purpose

Holds syndrome information for an exception taken to EL2.

Configuration

AArch64 System register ESR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HSR\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ESR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|------|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ISS2 | | | |
| EC | | | | | | IL | | ISS | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | |

ESR_EL2 is made UNKNOWN as a result of an exception return from EL2.

When an UNPREDICTABLE instruction is treated as UNDEFINED, and the exception is taken to EL2, the value of ESR_EL2 is UNKNOWN. The value written to ESR_EL2 must be consistent with a value that could be created as a result of an exception from the same Exception level that generated the exception as a result of a situation that is not UNPREDICTABLE at that Exception level, in order to avoid the possibility of a privilege violation.

Bits [63:37]

Reserved, RES0.

ISS2, bits [36:32]

When FEAT_LS64 is implemented:

If a memory access generated by an ST64BV or ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field holds register specifier, Xs.

For any other Data Abort, this field is RES0.

Otherwise:

Reserved, RES0.

EC, bits [31:26]

Exception Class. Indicates the reason for the exception that this register holds information about.

For each EC value, the table references a subsection that gives information about:

- The cause of the exception, for example the configuration required to enable the trap.

- The encoding of the associated ISS.

Possible values of the EC field are:

| EC | Meaning | ISS | Applies when |
|----------|---|--|---------------------------|
| 0b000000 | Unknown reason. | ISS encoding for exceptions with an unknown reason | |
| 0b000001 | Trapped WF* instruction execution. Conditional WF* instructions that fail their condition code check do not cause an exception. | ISS encoding for an exception from a WF* instruction | |
| 0b000011 | Trapped MCR or MRC access with (coproc==0b1111) that is not reported using EC 0b000000. | ISS encoding for an exception from an MCR or MRC access | When AArch32 is supported |
| 0b000100 | Trapped MCRR or MRRC access with (coproc==0b1111) that is not reported using EC 0b000000. | ISS encoding for an exception from an MCRR or MRRC access | When AArch32 is supported |
| 0b000101 | Trapped MCR or MRC access with (coproc==0b1110). | ISS encoding for an exception from an MCR or MRC access | When AArch32 is supported |
| 0b000110 | Trapped LDC or STC access. The only architected uses of these instruction are: <ul style="list-style-type: none"> An STC to write data to memory from DBGDTRRXint. An LDC to read data from memory to DBGDTRTXint. | ISS encoding for an exception from an LDC or STC instruction | When AArch32 is supported |
| 0b000111 | Access to SME, SVE, Advanced SIMD or floating-point functionality trapped by CPACR_EL1.FPEN , CPTR_EL2.FPEN , CPTR_EL2.TFP , or CPTR_EL3.TFP control. Excludes exceptions resulting from CPACR_EL1 when the value of HCR_EL2.TGE is 1, or because SVE or Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000 as described in 'The EC used to report an exception routed to EL2 because HCR_EL2.TGE is 1'. | ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality resulting from the FPEN and TFP traps | |

| | | | |
|----------|---|---|--------------------------------|
| 0b001000 | Trapped VMRS access, from ID group trap, that is not reported using EC 0b000111. | ISS encoding for an exception from an MCR or MRC access | When AArch32 is supported |
| 0b001001 | Trapped use of a Pointer authentication instruction because HCR_EL2.API == 0 SCR_EL3.API == 0 . | ISS encoding for an exception from a Pointer Authentication instruction when HCR_EL2.API == 0 SCR_EL3.API == 0 | When FEAT_PAuth is implemented |
| 0b001010 | Trapped execution of an LD64B, ST64B, ST64BV, or ST64BV0 instruction. | ISS encoding for an exception from an LD64B or ST64B* instruction | When FEAT_LS64 is implemented |
| 0b001100 | Trapped MRRC access with (coproc==0b1110). | ISS encoding for an exception from an MCRR or MRRC access | When AArch32 is supported |
| 0b001101 | Branch Target Exception. | ISS encoding for an exception from Branch Target Identification instruction | When FEAT_BTI is implemented |
| 0b001110 | Illegal Execution state. | ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault | |
| 0b010001 | SVC instruction execution in AArch32 state. This is reported in ESR_EL2 only when the exception is generated because the value of HCR_EL2.TGE is 1. | ISS encoding for an exception from HVC or SVC instruction execution | When AArch32 is supported |
| 0b010010 | HVC instruction execution in AArch32 state, when HVC is not disabled. | ISS encoding for an exception from HVC or SVC instruction execution | When AArch32 is supported |
| 0b010011 | SMC instruction execution in AArch32 state, when SMC is not disabled. This is reported in ESR_EL2 only when the exception is generated because the value of HCR_EL2.TSC is 1. | ISS encoding for an exception from SMC instruction execution in AArch32 state | When AArch32 is supported |
| 0b010101 | SVC instruction execution in AArch64 state. | ISS encoding for an exception from HVC or SVC instruction execution | When AArch64 is supported |
| 0b010110 | HVC instruction execution in AArch64 state, when HVC is not disabled. | ISS encoding for an exception from HVC or SVC instruction execution | When AArch64 is supported |

| | | | |
|----------|--|---|---|
| 0b010111 | SMC instruction execution in AArch64 state, when SMC is not disabled. This is reported in ESR_EL2 only when the exception is generated because the value of HCR_EL2.TSC is 1. | ISS encoding for an exception from SMC instruction execution in AArch64 state | When AArch64 is supported |
| 0b011000 | Trapped MSR, MRS or System instruction execution in AArch64 state, that is not reported using EC 0b000000, 0b000001 or 0b000111. This includes all instructions that cause exceptions that are part of the encoding space defined in 'System instruction class encoding overview', except for those exceptions reported using EC values 0b000000, 0b000001, or 0b000111. | ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state | When AArch64 is supported |
| 0b011001 | Access to SVE functionality trapped as a result of CPACR_EL1.ZEN , CPTR_EL2.ZEN , CPTR_EL2.TZ , or CPTR_EL3.EZ , that is not reported using EC 0b000000. | ISS encoding for an exception from an access to SVE functionality resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ | When FEAT_SVE is implemented |
| 0b011010 | Trapped ERET, ERETAA, or ERETAB instruction execution. | ISS encoding for an exception from an ERET, ERETAA, or ERETAB instruction | When FEAT_PAuth is implemented and FEAT_NV is implemented |
| 0b011011 | Exception from an access to a TSTART instruction at EL0 when SCTLR_EL1.TME0 == 0, EL0 when SCTLR_EL2.TME0 == 0, at EL1 when SCTLR_EL1.TME == 0, at EL2 when SCTLR_EL2.TME == 0 or at EL3 when SCTLR_EL3.TME == 0. | ISS encoding for an exception from a TSTART instruction | When FEAT_TME is implemented |
| 0b011100 | Exception from a Pointer Authentication instruction authentication failure | ISS encoding for an exception from a Pointer Authentication instruction | When FEAT_FPAC is implemented |

| | | | |
|----------|--|--|------------------------------|
| 0b011101 | Access to SME functionality trapped as a result of CPACR_EL1.SMEN , CPTR_EL2.SMEN , CPTR_EL2.TSM , CPTR_EL3.ESM , or an attempted execution of an instruction that is illegal because of the value of PSTATE.SM or PSTATE.ZA , that is not reported using EC 0b000000. | authentication failure ISS encoding for an exception due to SME functionality | When FEAT_SME is implemented |
| 0b011110 | Exception from a Granule Protection Check | ISS encoding for an exception from a Granule Protection Check | When FEAT_RME is implemented |
| 0b100000 | Instruction Abort from a lower Exception level. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions. | ISS encoding for an exception from an Instruction Abort | |
| 0b100001 | Instruction Abort taken without a change in Exception level. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions. | ISS encoding for an exception from an Instruction Abort | |
| 0b100010 | PC alignment fault exception. | ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault | |
| 0b100100 | Data Abort from a lower Exception level, excluding Data Aborts taken to EL2 as a result of accesses generated associated with VNCR_EL2 as part of nested virtualization support. These Data Aborts might be generated from Exception | ISS encoding for an exception from a Data Abort | |

| | | | |
|----------|---|---|--------------------------------------|
| | <p>levels in any Execution state. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.</p> | | |
| 0b100101 | <p>Data Abort without a change in Exception level, or Data Aborts taken to EL2 as a result of accesses generated associated with VNCR_EL2 as part of nested virtualization support. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.</p> | <p>ISS encoding for an exception from a Data Abort</p> | |
| 0b100110 | <p>SP alignment fault exception.</p> | <p>ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault</p> | |
| 0b100111 | <p>Memory Operation Exception.</p> | <p>ISS encoding for an exception from the Memory Copy and Memory Set instructions</p> | <p>When FEAT_MOPS is implemented</p> |
| 0b101000 | <p>Trapped floating-point exception taken from AArch32 state. This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of</p> | <p>ISS encoding for an exception from a trapped floating-point exception</p> | <p>When AArch32 is supported</p> |

| | | | |
|----------|--|---|---------------------------|
| | floating-point exceptions is IMPLEMENTATION DEFINED. | | |
| 0b101100 | Trapped floating-point exception taken from AArch64 state. This EC value is valid if the implementation supports trapping of floating-point exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED. | ISS encoding for an exception from a trapped floating-point exception | When AArch64 is supported |
| 0b101111 | SError interrupt. | ISS encoding for an SError interrupt | |
| 0b110000 | Breakpoint exception from a lower Exception level. | ISS encoding for an exception from a Breakpoint or Vector Catch debug exception | |
| 0b110001 | Breakpoint exception taken without a change in Exception level. | ISS encoding for an exception from a Breakpoint or Vector Catch debug exception | |
| 0b110010 | Software Step exception from a lower Exception level. | ISS encoding for an exception from a Software Step exception | |
| 0b110011 | Software Step exception taken without a change in Exception level. | ISS encoding for an exception from a Software Step exception | |
| 0b110100 | Watchpoint from a lower Exception level, excluding Watchpoint Exceptions taken to EL2 as a result of accesses generated associated with VNCR_EL2 as part of nested virtualization support. These Watchpoint Exceptions might be generated from Exception levels using any Execution state. | ISS encoding for an exception from a Watchpoint exception | |
| 0b110101 | Watchpoint exceptions without a change in Exception level, or Watchpoint exceptions taken to EL2 as a result of | ISS encoding for an exception from a Watchpoint exception | |

| | | | |
|----------|---|---|---------------------------|
| | accesses generated associated with VNCR_EL2 as part of nested virtualization support. | | |
| 0b111000 | BKPT instruction execution in AArch32 state. | ISS encoding for an exception from execution of a Breakpoint instruction | When AArch32 is supported |
| 0b111010 | Vector Catch exception from AArch32 state. The only case where a Vector Catch exception is taken to an Exception level that is using AArch64 is when the exception is routed to EL2 and EL2 is using AArch64. | ISS encoding for an exception from a Breakpoint or Vector Catch debug exception | When AArch32 is supported |
| 0b111100 | BRK instruction execution in AArch64 state. | ISS encoding for an exception from execution of a Breakpoint instruction | When AArch64 is supported |

All other EC values are reserved by Arm, and:

- Unused values in the range 0b000000 - 0b101100 (0x00 - 0x2C) are reserved for future use for synchronous exceptions.
- Unused values in the range 0b101101 - 0b111111 (0x2D - 0x3F) are reserved for future use, and might be used for synchronous or asynchronous exceptions.

The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [25]

Instruction Length for synchronous exceptions. Possible values of this bit are:

| IL | Meaning |
|-----|--|
| 0b0 | 16-bit instruction trapped. |
| 0b1 | 32-bit instruction trapped. This value is also used when the exception is one of the following: <ul style="list-style-type: none"> • An SError interrupt. • An Instruction Abort exception. • A PC alignment fault exception. • An SP alignment fault exception. • A Data Abort exception for which the value of the ISV bit is 0. • An Illegal Execution state exception. • Any debug exception except for Breakpoint instruction exceptions. For Breakpoint instruction exceptions, this bit has its standard meaning: <ul style="list-style-type: none"> ◦ 0b0: 16-bit T32 BKPT instruction. ◦ 0b1: 32-bit A32 BKPT instruction or A64 BRK instruction. • An exception reported using EC value 0b000000. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [24:0]

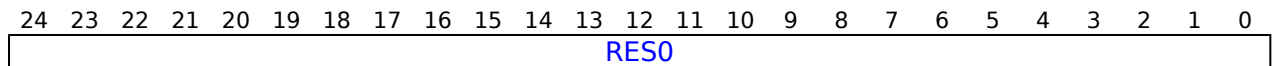
Instruction Specific Syndrome. Architecturally, this field can be defined independently for each defined Exception class. However, in practice, some ISS encodings are used for more than one Exception class.

Typically, an ISS encoding has a number of subfields. When an ISS subfield holds a register number, the value returned in that field is the AArch64 view of the register number.

For an exception taken from AArch32 state, see 'Mapping of the general-purpose registers between the Execution states'.

If the AArch32 register descriptor is 0b1111, then:

- If the instruction that generated the exception was not UNPREDICTABLE, the field takes the value 0b11111.
- If the instruction that generated the exception was UNPREDICTABLE, the field takes an UNKNOWN value that must be either:
 - The AArch64 view of the register number of a register that might have been used at the Exception level from which the exception was taken.
 - The value 0b11111.

ISS encoding for exceptions with an unknown reason**Bits [24:0]**

Reserved, RES0.

When an exception is reported using this EC code the IL field is set to 1.

This EC code is used for all exceptions that are not covered by any other EC value. This includes exceptions that are generated in the following situations:

- The attempted execution of an instruction bit pattern that has no allocated instruction or that is not accessible at the current Exception level and Security state, including:
 - A read access using a System register pattern that is not allocated for reads or that does not permit reads at the current Exception level and Security state.
 - A write access using a System register pattern that is not allocated for writes or that does not permit writes at the current Exception level and Security state.
 - Instruction encodings that are unallocated.
 - Instruction encodings for instructions or System registers that are not implemented in the implementation.
- In Debug state, the attempted execution of an instruction bit pattern that is not accessible in Debug state.
- In Non-debug state, the attempted execution of an instruction bit pattern that is not accessible in Non-debug state.
- In AArch32 state, attempted execution of a short vector floating-point instruction.
- In an implementation that does not include Advanced SIMD and floating-point functionality, an attempted access to Advanced SIMD or floating-point functionality under conditions where that access would be permitted if that functionality was present. This includes the attempted execution of an Advanced SIMD or floating-point instruction, and attempted accesses to Advanced SIMD and floating-point System registers.
- An exception generated because of the value of one of the [SCTLR_EL1](#).{ITD, SED, CP15BEN} control bits.
- Attempted execution of:
 - An HVC instruction when disabled by [HCR_EL2](#).HCD or [SCR_EL3](#).HCE.
 - An SMC instruction when disabled by [SCR_EL3](#).SMD.
 - An HLT instruction when disabled by [EDSCR](#).HDE.
- Attempted execution of an MSR or MRS instruction to access [SP_EL0](#) when the value of [SPSel](#).SP is 0.
- Attempted execution of an MSR or MRS instruction using a [_EL12](#) register name when [HCR_EL2](#).E2H == 0.
- Attempted execution, in Debug state, of:
 - A DCPS1 instruction when the value of [HCR_EL2](#).TGE is 1 and EL2 is disabled or not implemented in the current Security state.
 - A DCPS2 instruction from EL1 or EL0 when EL2 is disabled or not implemented in the current Security state.

- A DCPS3 instruction when the value of [EDSCR.SDD](#) is 1, or when EL3 is not implemented.
- When EL3 is using AArch64, attempted execution from Secure EL1 of an SRS instruction using R13_mon. See 'Traps to EL3 of Secure monitor functionality from Secure EL1 using AArch32'.
- In Debug state when the value of [EDSCR.SDD](#) is 1, the attempted execution at EL2, EL1, or EL0 of an instruction that is configured to trap to EL3.
- In AArch32 state, the attempted execution of an MRS (banked register) or an MSR (banked register) instruction to SPSR_mon, SP_mon, or LR_mon.
- An exception that is taken to EL2 because the value of [HCR_EL2.TGE](#) is 1 that, if the value of [HCR_EL2.TGE](#) was 0 would have been reported with an ESR_ELx.EC value of 0b000111.
- In Non-transactional state, attempted execution of a TCOMMIT instruction.

ISS encoding for an exception from a WF* instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|----|----|----|----|----|----|----|---|---|------|---|----|----|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | RES0 | | | | | | | | | | RN | | | RES0 | | RV | TI | | | |

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:10]

Reserved, RES0.

RN, bits [9:5]

When FEAT_WFxT is implemented:

Register Number. Indicates the register number supplied for a WFET or WFIT instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [4:3]

Reserved, RES0.

RV, bit [2]

When FEAT_WFxT is implemented:

Register field Valid.

If TI[1] == 1, then this field indicates whether RN holds a valid register number for the register argument to the trapped WFET or WFIT instruction.

| RV | Meaning |
|-----|-------------------------|
| 0b0 | Register field invalid. |
| 0b1 | Register field valid. |

If TI[1] == 0, then this field is RES0.

This field is set to 1 on a trap on WFET or WFIT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TI, bits [1:0]

Trapped instruction. Possible values of this bit are:

| TI | Meaning | Applies when |
|------|---------------|-------------------------------|
| 0b00 | WFI trapped. | |
| 0b01 | WFE trapped. | |
| 0b10 | WFIT trapped. | When FEAT_WFxT is implemented |
| 0b11 | WFET trapped. | When FEAT_WFxT is implemented |

When FEAT_WFxT is implemented, this is a two bit field as shown. Otherwise, bit[1] is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following fields describe configuration settings for generating this exception:

- [SCTLR_EL1](#).{nTWE, nTWI}.
- [HCR_EL2](#).{TWE, TWI}.
- [SCR_EL3](#).{TWE, TWI}.

ISS encoding for an exception from an MCR or MRC access

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|------|----|-----|----|----|----|----|----|---|---|-----|---|---|---|-----------|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | Opc2 | | Opc1 | | CRn | | | | Rt | | | | CRm | | | | Direction | | | |

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc2, bits [19:17]

The Opc2 value from the issued instruction.

For a trapped VMRS access, holds the value 0b000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [16:14]

The Opc1 value from the issued instruction.

For a trapped VMRS access, holds the value 0b111.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

For a trapped VMRS access, holds the reg field from the VMRS instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

If the Rt value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b1111.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
 - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
 - The value 0b1111.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

For a trapped VMRS access, holds the value 0b0000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|---|
| 0b0 | Write to System register space. MCR instruction. |
| 0b1 | Read from System register space. MRC or VMRS instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000011:

- [CNTKCTL_EL1](#).{ELOPTEN, EL0VTEN, ELOPCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [PMUSERENR_EL0](#).{ER, CR, SW, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [AMUSERENR_EL0](#).EN, for accesses to Activity Monitors registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [HCR_EL2](#).{TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).TLB, for execution of TLB maintenance instructions at EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).{TSW, TPC, TPU} for execution of cache maintenance instructions at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).TACR, for accesses to the Auxiliary Control Register at EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).TIDCP, for accesses to lockdown, DMA, and TCM operations at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).{TID1, TID2, TID3}, for accesses to ID registers at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [CPTR_EL2](#).TCPAC, for accesses to [CPACR_EL1](#) or [CPACR](#) using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HSTR_EL2](#).T<n>, for accesses to System registers using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [CNTHCTL_EL2](#).EL1PCEN, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [MDCR_EL2](#).{TPM, TPMCR}, for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [CPTR_EL2](#).TAM, for accesses to Activity Monitors registers from EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [CPTR_EL3](#).TCPAC, for accesses to [CPACR](#) from EL1 and EL2, and accesses to [HCPTR](#) from EL2 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL3.
- [MDCR_EL3](#).TPM, for accesses to Performance Monitor registers from EL0, EL1 and EL2 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL3.
- [CPTR_EL3](#).TAM, for accesses to Activity Monitors registers from EL0, EL1 and EL2 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL3.
- For information on other traps using EC value 0b000011, see 'Traps to EL3 of Secure monitor functionality from Secure EL1 using AArch32'.
- If FEAT_FGT is implemented, MCR or MRC access to some registers at EL0, trapped to EL2.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000101:

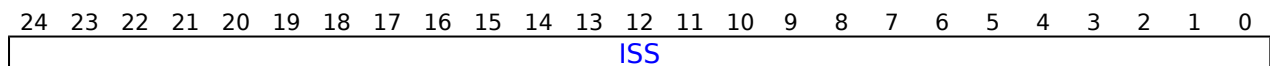
- [CPACR_EL1](#).TTA for accesses to trace registers, MCR or MRC access (coproc == 0b1110) trapped to EL1 or EL2.
- [MDSCR_EL1](#).TDCC, for accesses to the Debug Communications Channel (DCC) registers at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1110) trapped to EL1 or EL2.
- If FEAT_FGT is implemented, [MDSCR_EL2](#).TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and [MDSCR_EL3](#).TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.
- [HCR_EL2](#).TID0, for accesses to the [JIDR](#) register in the ID group 0 at EL0 and EL1 using AArch32, MRC access (coproc == 0b1110) trapped to EL2.
- [CPTR_EL2](#).TTA, for accesses to trace registers using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- [MDCR_EL2](#).TDRA, for accesses to Debug ROM registers [DBGDRAR](#) and [DBGDSAR](#) using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- [MDCR_EL2](#).TDOSA, for accesses to powerdown debug registers, using AArch32 state, MCR or MRC access (coproc == 0b1110) trapped to EL2.

- [MDCR_EL2](#).TDA, for accesses to other debug registers, using AArch32 state, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- [CPTR_EL3](#).TTA, for accesses to trace registers using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL3.
- [MDCR_EL3](#).TDOSA, for accesses to powerdown debug registers using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL3.
- [MDCR_EL3](#).TDA, for accesses to other debug registers, using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL3.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b001000:

- [HCR_EL2](#).TID0, for accesses to the [FPSID](#) register in ID group 0 at EL1 using AArch32 state, VMRS access trapped to EL2.
- [HCR_EL2](#).TID3, for accesses to registers in ID group 3 including [MVFR0](#), [MVFR1](#) and [MVFR2](#), VMRS access trapped to EL2.

ISS encoding for an exception from an LD64B or ST64B* instruction

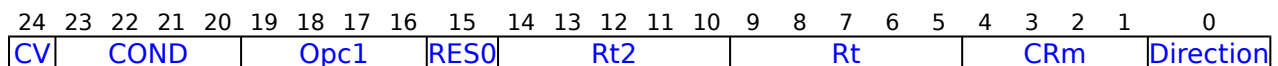


ISS, bits [24:0]

| ISS | Meaning |
|----------------------------------|-------------------------------------|
| 0b000000000000000000000000000000 | ST64BV instruction trapped. |
| 0b000000000000000000000000000001 | ST64BV0 instruction trapped. |
| 0b000000000000000000000000000010 | LD64B or ST64B instruction trapped. |

All other values are reserved.

ISS encoding for an exception from an MCRR or MRRC access



CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [19:16]

The Opc1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

Rt2, bits [14:10]

The Rt2 value from the issued instruction, the second general-purpose register used for the transfer.

If the Rt2 value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt2 value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b1111.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
 - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
 - The value 0b1111.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the first general-purpose register used for the transfer.

If the Rt value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b11111.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
 - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
 - The value 0b11111.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|--|
| 0b0 | Write to System register space. MCRR instruction. |
| 0b1 | Read from System register space. MRRC instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000100:

- [CNTKCTL_EL1](#).{ELOPTEN, EL0VTEN, EL0PCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [PMUSERENR_EL0](#).{CR, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [AMUSERENR_EL0](#).{EN}, for accesses to Activity Monitors registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [HCR_EL2](#).{TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [HSTR_EL2](#).T<n>, for accesses to System registers using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [CNTHCTL_EL2](#).{EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [MDCR_EL2](#).{TPM, TPMCR}, for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [CPTR_EL2](#).TAM, for accesses to Activity Monitors registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [MDCR_EL3](#).TPM, for accesses to Performance Monitor registers from EL0, EL1 and EL2 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL3.
- [CPTR_EL3](#).TAM, for accesses to Activity Monitors registers from EL0, EL1 and EL2 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL3.

- If FEAT_FGT is implemented, [HDFGRTR_EL2](#).PMCCNTR_EL0 for MRRC access and [HDFGWTR_EL2](#).PMCCNTR_EL0 for MCRR access to [PMCCNTR](#) at EL0, trapped to EL2.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b001100:

- [MDSCR_EL1](#).TDCC, for accesses to the Debug ROM registers [DBGDSAR](#) and [DBGDRAR](#) at EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1110) trapped to EL1 or EL2.
- [MDCR_EL2](#).TDRA, for accesses to Debug ROM registers [DBGDRAR](#) and [DBGDSAR](#) using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL2.
- [MDCR_EL3](#).TDA, for accesses to debug registers, using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL3.
- [CPACR_EL1](#).TTA for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL1 or EL2.
- [CPTR_EL2](#).TTA, for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL2.
- [CPTR_EL3](#).TTA, for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL3.

Note

If the Armv8-A architecture is implemented with an ETMv4 implementation, MCRR and MRRC accesses to trace registers are UNDEFINED and the resulting exception is higher priority than an exception due to these traps.

ISS encoding for an exception from an LDC or STC instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|----|----|----|----|------|----|---|---|---|--------|----|---|-----------|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | imm8 | | | | | | | | RES0 | Rn | | | | Offset | AM | | Direction | | | |

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to 0b1110, the value for unconditional.
- With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

imm8, bits [19:12]

The immediate value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

Rn, bits [9:5]

The Rn value from the issued instruction, the general-purpose register used for the transfer.

If the Rn value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rn value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b1111.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
 - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
 - The value 0b1111.

See 'Mapping of the general-purpose registers between the Execution states'.

This field is valid only when AM[2] is 0, indicating an immediate form of the LDC or STC instruction. When AM[2] is 1, indicating a literal form of the LDC or STC instruction, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Offset, bit [4]

Indicates whether the offset is added or subtracted:

| Offset | Meaning |
|--------|------------------|
| 0b0 | Subtract offset. |
| 0b1 | Add offset. |

This bit corresponds to the U bit in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AM, bits [3:1]

Addressing mode. The permitted values of this field are:

| AM | Meaning |
|-------|---|
| 0b000 | Immediate unindexed. |
| 0b001 | Immediate post-indexed. |
| 0b010 | Immediate offset. |
| 0b011 | Immediate pre-indexed. |
| 0b100 | For a trapped STC instruction or a trapped T32 LDC instruction this encoding is reserved. |
| 0b110 | For a trapped STC instruction, this encoding is reserved. |

The values 0b101 and 0b111 are reserved. The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in System and memory-mapped registers and translation table entries'.

Bit [2] in this subfield indicates the instruction form, immediate or literal.

Bits [1:0] in this subfield correspond to the bits {P, W} in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|------------------------------------|
| 0b0 | Write to memory. STC instruction. |
| 0b1 | Read from memory. LDC instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following fields describe the configuration settings for the traps that are reported using EC value 0b000110:

- [MDSCR_EL1](#).TDCC, for accesses using AArch32 state, LDC access to [DBGDTRTXint](#) or STC access to [DBGDTRRXint](#) trapped to EL1 or EL2.
- [MDCR_EL2](#).TDA, for accesses using AArch32 state, LDC access to [DBGDTRTXint](#) or STC access to [DBGDTRRXint](#) MCR or MRC access trapped to EL2.
- [MDCR_EL3](#).TDA, for accesses using AArch32 state, LDC access to [DBGDTRTXint](#) or STC access to [DBGDTRRXint](#) MCR or MRC access trapped to EL3.
- If FEAT_FGT is implemented, [MDCR_EL2](#).TDCC for LDC and STC accesses to the DCC registers at EL0 and EL1 trapped to EL2, and [MDCR_EL3](#).TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.

ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from the FPEN and TFP traps

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | RES0 | | | | | | | | | | | | | | | | | | | |

The accesses covered by this trap include:

- Execution of SVE or Advanced SIMD and floating-point instructions.
- Accesses to the Advanced SIMD and floating-point System registers.
- Execution of SME instructions.

For an implementation that does not include either SVE or support for Advanced SIMD and floating-point, the exception is reported using the EC value 0b000000.

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

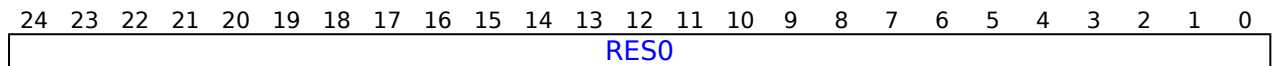
Bits [19:0]

Reserved, RES0.

The following fields describe the configuration settings for the traps that are reported using EC value 0b000111:

- [CPACR_EL1](#).FPEN, for accesses to SIMD and floating-point registers trapped to EL1.
- [CPTR_EL2](#).FPEN and [CPTR_EL2](#).TFP, for accesses to SIMD and floating-point registers trapped to EL2.
- [CPTR_EL3](#).TFP, for accesses to SIMD and floating-point registers trapped to EL3.

ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ



The accesses covered by this trap include:

- Execution of SVE instructions when the PE is not in Streaming SVE mode.
- Accesses to the SVE System registers, ZCR_ELx.

For an implementation that does not include SVE, the exception is reported using the EC value 0b000000.

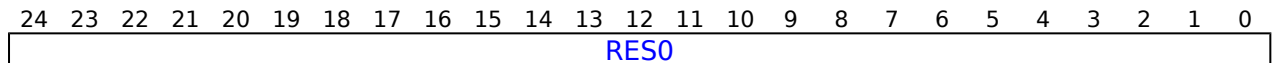
Bits [24:0]

Reserved, RES0.

The following fields describe the configuration settings for the traps that are reported using EC value 0b011001:

- [CPACR_EL1.ZEN](#), for execution of SVE instructions and accesses to SVE registers at EL0 or EL1, trapped to EL1.
- [CPTR_EL2.ZEN](#) and [CPTR_EL2.TZ](#), for execution of SVE instructions and accesses to SVE registers at EL0, EL1, or EL2, trapped to EL2.
- [CPTR_EL3.EZ](#), for execution of SVE instructions and accesses to SVE registers from all Exception levels, trapped to EL3.

ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault



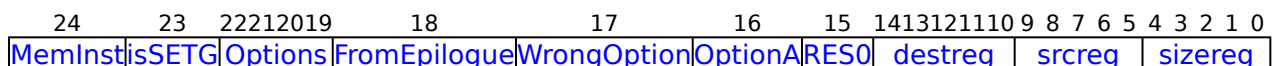
Bits [24:0]

Reserved, RES0.

There are no configuration settings for generating Illegal Execution state exceptions and PC alignment fault exceptions. For more information about these exceptions, see 'The Illegal Execution state exception' and 'PC alignment checking'.

'SP alignment checking' describes the configuration settings for generating SP alignment fault exceptions.

ISS encoding for an exception from the Memory Copy and Memory Set instructions



MemInst, bit [24]

Indicates the memory instruction class causing the exception.

| MemInst | Meaning |
|---------|--|
| 0b0 | CPYFE*, CPYFM*, CPYE*, and CPYM* instructions. |
| 0b1 | SETE*, SETM*, SETGE*, and SETGM* instructions. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

isSETG, bit [23]

Indicates whether the instruction belongs to SETGM* or SETGE* class of instruction.

| isSETG | Meaning |
|---------------|-------------------------------------|
| 0b0 | Not a SETGM* or SETGE* instruction. |
| 0b1 | SETGM* or SETGE* instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Options, bits [22:19]

Options : the Options field of the instruction.

For Memory Copy instructions, bits[22:19] forms the Options field, which holds the bits[15:12] of the instruction.

For Memory Set instructions:

- Bits[22:21] are RES0.
- Bits[20:19] form the Options field, which holds the bits[13:12] of the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FromEpilogue, bit [18]

Indicates whether the instruction belongs to the epilogue class of Memory Copy or Memory Set instructions.

| FromEpilogue | Meaning |
|---------------------|--|
| 0b0 | Not an epilogue instruction. |
| 0b1 | CPYE*, CPYFE*, SETE*, or SETGE* instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WrongOption, bit [17]

Algorithm option.

| WrongOption | Meaning |
|--------------------|-----------------------|
| 0b0 | WrongOption is false. |
| 0b1 | WrongOption is true. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OptionA, bit [16]

Algorithm type indicated by the PSTATE.C bit.

| OptionA | Meaning |
|----------------|-------------------------------------|
| 0b0 | OptionB indicated by PSTATE.C is 0. |
| 0b1 | OptionA indicated by PSTATE.C is 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

destreg, bits [14:10]

The destination register value from the issued instruction, containing the destination address.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

srcreg, bits [9:5]

The source register value from the issued instruction, containing either the source address or the source data.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

sizereg, bits [4:0]

The size register value from the issued instruction, containing the number of bytes to be transferred or set.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from HVC or SVC instruction execution

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | imm16 | | | | | | | | | | | | | | | |

Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the HVC or SVC instruction.

For an HVC instruction, and for an A64 SVC instruction, this is the value of the imm16 field of the issued instruction.

For an A32 or T32 SVC instruction:

- If the instruction is unconditional, then:
 - For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
 - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- If the instruction is conditional, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

In AArch32 state, the HVC instruction is unconditional, and a conditional SVC instruction generates an exception only if it passes its condition code check. Therefore, the syndrome information for these exceptions does not require conditionality information.

For T32 and A32 instructions, see 'SVC' and 'HVC'.

For A64 instructions, see 'SVC' and 'HVC'.

If FEAT_FGT is implemented, [HFGITR_EL2](#).{SVC_EL1, SVC_EL0} control fine-grained traps on SVC execution.

ISS encoding for an exception from SMC instruction execution in AArch32 state

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|-------------|----|----|----|----|----|----|----|----|----|------|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | CCKNOWNPASS | | | | | | | | | | RES0 | | | | | | | | | |

For an SMC instruction that completes normally and generates an exception that is taken to EL3, the ISS encoding is RES0.

For an SMC instruction that is trapped to EL2 from EL1 because [HCR_EL2](#).TSC is 1, the ISS encoding is as shown in the diagram.

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CCKNOWNPASS, bit [19]

Indicates whether the instruction might have failed its condition code check.

| CCKNOWNPASS | Meaning |
|-------------|--|
| 0b0 | The instruction was unconditional, or was conditional and passed its condition code check. |
| 0b1 | The instruction was conditional, and might have failed its condition code check. |

Note

In an implementation in which an SMC instruction that fails its code check is not trapped, this field can always return the value 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [18:0]

Reserved, RES0.

[HCR_EL2.TSC](#) describes the configuration settings for trapping SMC instructions to EL2.

'System calls' describes the case where these exceptions are trapped to EL3.

ISS encoding for an exception from SMC instruction execution in AArch64 state

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | imm16 | | | | | | | | | | | | | | | |

Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the issued SMC instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The value of ISS[24:0] described here is used both:

- When an SMC instruction is trapped from EL1 modes.
- When an SMC instruction is not trapped, so completes normally and generates an exception that is taken to EL3.

[HCR_EL2](#).TSC describes the configuration settings for trapping SMC from EL1 modes.

'System calls' describes the case where these exceptions are trapped to EL3.

ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-----|----|-----|----|-----|----|-----|----|----|----|----|---|---|---|-----|---|---|-----------|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | Op0 | | Op2 | | Op1 | | CRn | | | | Rt | | | | CRm | | | Direction | | | |

Bits [24:22]

Reserved, RES0.

Op0, bits [21:20]

The Op0 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op2, bits [19:17]

The Op2 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op1, bits [16:14]

The Op1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|---|
| 0b0 | Write access, including MSR instructions. |
| 0b1 | Read access, including MRS instructions. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For exceptions caused by System instructions, see 'System instructions' subsection of 'Branches, exception generating and System instructions' for the encoding values returned by an instruction.

The following fields describe configuration settings for generating the exception that is reported using EC value 0b011000:

- [SCTLR_EL1](#).UCI, for execution of cache maintenance instructions using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [SCTLR_EL1](#).UCT, for accesses to [CTR_EL0](#) using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [SCTLR_EL1](#).DZE, for execution of DC ZVA instructions using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [SCTLR_EL1](#).UMA, for accesses to the PSTATE interrupt masks using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [CPACR_EL1](#).TTA, for accesses to the trace registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [MDSCR_EL1](#).TDCC, for accesses to the Debug Communications Channel (DCC) registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- If FEAT_FGT is implemented, [MDCR_EL2](#).TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and [MDCR_EL3](#).TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.
- [CNTKCTL_EL1](#).{ELOPTEN, EL0VTEN, EL0PCTEN, EL0VCTEN} accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [PMUSERENR_EL0](#).{ER, CR, SW, EN}, for accesses to the Performance Monitor registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [AMUSERENR_EL0](#).EN, for accesses to Activity Monitors registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [HCR_EL2](#).{TRVM, TVM}, for accesses to virtual memory control registers using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).TDZ, for execution of DC ZVA instructions using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).TLB, for execution of TLB maintenance instructions using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).{TSW, TPC, TPU}, for execution of cache maintenance instructions using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).TACR, for accesses to the Auxiliary Control Register, [ACTLR_EL1](#), using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).TIDCP, for accesses to lockdown, DMA, and TCM operations using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).{TID1, TID2, TID3}, for accesses to ID group 1, ID group 2 or ID group 3 registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [CPTR_EL2](#).TCPAC, for accesses to [CPACR_EL1](#), using AArch64 state, MSR or MRS access trapped to EL2.
- [CPTR_EL2](#).TTA, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).TTRF, for accesses to the trace filter control register, [TRFCR_EL1](#), using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).TDRA, for accesses to Debug ROM registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).TDOSA, for accesses to powerdown debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- [CNTHCTL_EL2](#).{EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).TDA, for accesses to debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).{TPM, TPMCR}, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL2.

- [CPTR_EL2](#).TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).APK, for accesses to Pointer authentication key registers. using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).{NV, NV1}, for Nested virtualization register access, using AArch64 state, MSR or MRS access, trapped to EL2.
- [HCR_EL2](#).AT, for execution of AT S1E* instructions, using AArch64 state, MSR or MRS access, trapped to EL2.
- [HCR_EL2](#).{TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access, trapped to EL2.
- [SCR_EL3](#).APK, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [SCR_EL3](#).ST, for accesses to the Counter-timer Physical Secure timer registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [SCR_EL3](#).{TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [CPTR_EL3](#).TCPAC, for accesses to [CPTR_EL2](#) and [CPACR_EL1](#) using AArch64 state, MSR or MRS access trapped to EL3.
- [CPTR_EL3](#).TTA, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [MDCR_EL3](#).TTRF, for accesses to the trace filter control registers, [TRFCR_EL1](#) and [TRFCR_EL2](#), using AArch64 state, MSR or MRS access trapped to EL3.
- [MDCR_EL3](#).TDA, for accesses to debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [MDCR_EL3](#).TDOSA, for accesses to powerdown debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [MDCR_EL3](#).TPM, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [CPTR_EL3](#).TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access, trapped to EL3.
- If FEAT_EVT is implemented, the following registers control traps for EL1 and EL0 Cache controls that use this EC value:
 - [HCR_EL2](#).{TTLBOS, TTLBIS, TICAB, TOCU, TID4}.
 - [HCR2](#).{TTLBIS, TICAB, TOCU, TID4}.
- If FEAT_FGT is implemented:
 - [SCR_EL3](#).FGTEn, for accesses to the fine-grained trap registers, MSR or MRS access at EL2 trapped to EL3.
 - [HFGTR_EL2](#) for reads and [HFGWTR_EL2](#) for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 trapped to EL2.
 - [HFGTR_EL2](#) for execution of system instructions, MSR or MRS access trapped to EL2
 - [HDFGTR_EL2](#) for reads and [HDFGWTR_EL2](#) for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 state trapped to EL2.
 - [HAFGTR_EL2](#) for reads of Activity Monitor counters, using AArch64 state, MRS access at EL0 and EL1 trapped to EL2.
- If FEAT_RNG_TRAP is implemented:
 - [SCR_EL3](#).TRNDR for reads of [RNDR](#) and [RNDRRS](#) using AArch64 state, MRS access trapped to EL3.
- If FEAT_SME is implemented:
 - [CPTR_EL3](#).ESM, for MSR or MRS accesses to [SMPRI_EL1](#) at EL1, EL2, and EL3, trapped to EL3.
 - [CPTR_EL3](#).ESM, for MSR or MRS accesses to [SMPRMAP_EL2](#) at EL2 and EL3, trapped to EL3.
 - [SCTLR_EL1](#).EnTP2, for MSR or MRS accesses to [TPIDR2_EL0](#) at EL0, trapped to EL1 or EL2.
 - [SCTLR_EL2](#).EnTP2, for MSR or MRS accesses to [TPIDR2_EL0](#) at EL0, trapped to EL2.
 - [SCR_EL3](#).EnTP2, for MSR or MRS accesses to [TPIDR2_EL0](#) at EL0, EL1, and EL2, trapped to EL3.
- If FEAT_NMI is implemented, [HCRX_EL2](#).TALLINT, for MSR writes of [ALLINT](#) at EL1, trapped to EL2.

ISS encoding for an IMPLEMENTATION DEFINED exception to EL3

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [24:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from an Instruction Abort

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|-----|-----|----|------|-------|------|------|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | SET | FnV | EA | RES0 | S1PTW | RES0 | IFSC | | | | | | |

Bits [24:13]

Reserved, RES0.

SET, bits [12:11]

When FEAT_RAS is implemented:

Synchronous Error Type. When IFSC is 0b010000, describes the PE error state after taking the Instruction Abort exception.

| SET | Meaning |
|------|--------------------------|
| 0b00 | Recoverable state (UER). |
| 0b10 | Uncontainable (UC). |
| 0b11 | Restartable state (UEO). |

All other values are reserved.

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in a PE state that is not recoverable.

This field is valid only if the IFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|---|
| 0b0 | FAR is valid. |
| 0b1 | FAR is not valid, and holds an UNKNOWN value. |

This field is valid only if the IFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

| S1PTW | Meaning |
|--------------|---|
| 0b0 | Fault not on a stage 2 translation for a stage 1 translation table walk. |
| 0b1 | Fault on the stage 2 translation of an access for a stage 1 translation table walk. |

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code.

| IFSC | Meaning | Applies when |
|----------|---|--|
| 0b000000 | Address size fault, level 0 of translation or translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000100 | Translation fault, level 0. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001000 | Access flag fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001100 | Permission fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010000 | Synchronous External abort, not on translation table walk or hardware update of translation table. | |
| 0b010011 | Synchronous External abort on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented |
| 0b010100 | Synchronous External abort on translation table walk or hardware update of translation table, level 0. | |
| 0b010101 | Synchronous External abort on translation table walk or hardware update of translation table, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk or hardware update of translation table, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk or hardware update of translation table, level 3. | |
| 0b011000 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b011011 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented and FEAT_RAS is not implemented |
| 0b011100 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0. | When FEAT_RAS is not implemented |
| 0b011101 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1. | When FEAT_RAS is not implemented |
| 0b011110 | Synchronous parity or ECC error on memory access on translation table walk or | When FEAT_RAS is |

| | | |
|----------|---|---|
| 0b011111 | hardware update of translation table, level 2. Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3. | not implemented When FEAT_RAS is not implemented |
| 0b100011 | Granule Protection Fault on translation table walk or hardware update of translation table, level -1. | When FEAT_RME is implemented and FEAT_LPA2 is implemented |
| 0b100100 | Granule Protection Fault on translation table walk or hardware update of translation table, level 0. | When FEAT_RME is implemented |
| 0b100101 | Granule Protection Fault on translation table walk or hardware update of translation table, level 1. | When FEAT_RME is implemented |
| 0b100110 | Granule Protection Fault on translation table walk or hardware update of translation table, level 2. | When FEAT_RME is implemented |
| 0b100111 | Granule Protection Fault on translation table walk or hardware update of translation table, level 3. | When FEAT_RME is implemented |
| 0b101000 | Granule Protection Fault, not on translation table walk or hardware update of translation table. | When FEAT_RME is implemented |
| 0b101001 | Address size fault, level -1. | When FEAT_LPA2 is implemented |
| 0b101011 | Translation fault, level -1. | When FEAT_LPA2 is implemented |
| 0b110000 | TLB conflict abort. | |
| 0b110001 | Unsupported atomic hardware update fault. | When FEAT_HAFDBS is implemented |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults'.

Note

Because Access flag faults and Permission faults can result only from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception due to SME functionality

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|------|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | SMTc | | |

The accesses covered by this trap include:

- Execution of SME instructions.
- Execution of SVE and Advanced SIMD instructions, when the PE is in Streaming SVE mode.
- Direct accesses of [SVCR](#), [SMCR_EL1](#), [SMCR_EL2](#), [SMCR_EL3](#).

Bits [24:3]

Reserved, RES0.

SMTC, bits [2:0]

SME Trap Code. Identifies the reason for instruction trapping.

| SMTC | Meaning |
|-------|--|
| 0b000 | Access to SME functionality trapped as a result of CPACR_EL1 .SMEN, CPTR_EL2 .SMEN, CPTR_EL2 .TSM, or CPTR_EL3 .ESM, that is not reported using EC 0b000000. |
| 0b001 | Advanced SIMD, SVE, or SVE2 instruction trapped because PSTATE.SM is 1. |
| 0b010 | SME instruction trapped because PSTATE.SM is 0. |
| 0b011 | SME instruction trapped because PSTATE.ZA is 0. |

All other values are reserved.

The following fields describe the configuration settings for the traps that are reported using the EC value 0b011101:

- [CPACR_EL1](#).SMEN, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access [SVCR](#) and [SMCR_EL1](#) System registers at EL1 and EL0, trapped to EL1 or EL2.
- [CPTR_EL2](#).SMEN and [CPTR_EL2](#).TSM, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access [SVCR](#), [SMCR_EL1](#), [SMCR_EL2](#) at EL2, EL1, or EL0, trapped to EL2.
- [CPTR_EL3](#).ESM, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access [SVCR](#), [SMCR_EL1](#), [SMCR_EL2](#), [SMCR_EL3](#) from all Exception levels and any Security state, trapped to EL3.

ISS encoding for an exception from a Granule Protection Check

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|-------|----|-----|-------|----|----|------|----|------|------|----|-------|-----|------|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | S2PTW | | InD | GPCSC | | | VNCR | | RES0 | RES0 | CM | S1PTW | WnR | xFSC | | | | | | | | |

Bits [24:22]

Reserved, RES0.

S2PTW, bit [21]

Indicates whether the Granule Protection Check exception was on an access made for a stage 2 translation table walk.

| S2PTW | Meaning |
|-------|--|
| 0b0 | Fault not on a stage 2 translation table walk. |
| 0b1 | Fault on a stage 2 translation table walk. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

InD, bit [20]

Indicates whether the Granule Protection Check exception was on an instruction or data access.

| InD | Meaning |
|-----|---------------------|
| 0b0 | Data access. |
| 0b1 | Instruction access. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GPCSC, bits [19:14]

Granule Protection Check Status Code.

| GPCSC | Meaning |
|----------|---|
| 0b000000 | GPT address size fault at level 0. |
| 0b000001 | GPT address size fault at level 1. |
| 0b000100 | GPT walk fault at level 0. |
| 0b000101 | GPT walk fault at level 1. |
| 0b001100 | Granule protection fault at level 0. |
| 0b001101 | Granule protection fault at level 1. |
| 0b010100 | Synchronous External abort on GPT fetch at level 0. |
| 0b010101 | Synchronous External abort on GPT fetch at level 1. |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VNCR, bit [13]

When FEAT_NV2 is implemented:

Indicates that the fault came from use of [VNCR_EL2](#) register by EL1 code.

| VNCR | Meaning |
|------|--|
| 0b0 | The fault was not generated by the use of VNCR_EL2 , by an MRS or MSR instruction executed at EL1. |
| 0b1 | The fault was generated by the use of VNCR_EL2 , by an MRS or MSR instruction executed at EL1. |

This field is 0 in ESR_EL1.

When InD is '1', this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [12:11]

Reserved, RES0.

Bits [10:9]

Reserved, RES0.

CM, bit [8]

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:

| CM | Meaning |
|-----------|--|
| 0b0 | The Data Abort was not generated by the execution of one of the System instructions identified in the description of value 1. |
| 0b1 | The Data Abort was generated by either the execution of a cache maintenance instruction or by a synchronous fault on the execution of an address translation instruction. The DC ZVA , DC GVA , and DC GZVA instructions are not classified as cache maintenance instructions, and therefore their execution cannot cause this field to be set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

Indicates whether the Granule Protection Check exception was on an access for stage 2 translation for a stage 1 translation table walk:

| S1PTW | Meaning |
|--------------|---|
| 0b0 | Fault not on a stage 2 translation for a stage 1 translation table walk. |
| 0b1 | Fault on the stage 2 translation of an access for a stage 1 translation table walk. |

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

| WnR | Meaning |
|------------|--|
| 0b0 | Abort caused by an instruction reading from a memory location. |
| 0b1 | Abort caused by an instruction writing to a memory location. |

When InD is '1', this field is RES0.

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is UNKNOWN for:

- An External abort on an Atomic access.
- A fault reported using a DFSC value of 0b110101 or 0b110001, indicating an unsupported Exclusive or atomic access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

xFSC, bits [5:0]

Instruction or Data Fault Status Code.

| xFSC | Meaning | Applies when |
|-------------|---|---|
| 0b100011 | Granule Protection Fault on translation table walk or hardware update of translation table, level -1. | When FEAT_RME is implemented and FEAT_LPA2 is implemented |
| 0b100100 | Granule Protection Fault on translation table walk or hardware update of translation table, level 0. | When FEAT_RME is implemented |
| 0b100101 | Granule Protection Fault on translation table walk or hardware update of translation table, level 1. | When FEAT_RME is implemented |
| 0b100110 | Granule Protection Fault on translation table walk or hardware update of translation table, level 2. | When FEAT_RME is implemented |
| 0b100111 | Granule Protection Fault on translation table walk or hardware update of translation table, level 3. | When FEAT_RME is implemented |
| 0b101000 | Granule Protection Fault, not on translation table walk or hardware update of translation table. | When FEAT_RME is implemented |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults'.

Note

Because Access flag faults and Permission faults can result only from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from a Data Abort

| | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|-----|-----|----|----|----|-----|----|----|----|----|------|-------------|-----|----|----|-------|-----|---|---|---|---|---|------|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ISV | SAS | SSE | | | | SRT | | | SF | AR | VNCR | Bits[12:11] | FnV | EA | CM | S1PTW | WnR | | | | | | DFSC | |

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV or ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this ISS encoding includes ISS2, bits[36:32].

ISV, bit [24]

Instruction Syndrome Valid. Indicates whether the syndrome information in ISS[23:14] is valid.

| ISV | Meaning |
|-----|---|
| 0b0 | No valid instruction syndrome. ISS[23:14] are RES0. |
| 0b1 | ISS[23:14] hold a valid instruction syndrome. |

In ESR_EL2, ISV is 1 when FEAT_LS64 is implemented and a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

For other faults reported in ESR_EL2, ISV is 0 except for the following stage 2 aborts:

- AArch64 loads and stores of a single general-purpose register (including the register specified with 0b11111, including those with Acquire/Release semantics, but excluding Load Exclusive or Store Exclusive and excluding those with writeback).
- AArch32 instructions where the instruction:
 - Is an LDR, LDA, LDRT, LDRSH, LDRSHT, LDRH, LDAH, LDRHT, LDRSB, LDRSBT, LDRB, LDAB, LDRBT, STR, STL, STRT, STRH, STLH, STRHT, STRB, STLB, or STRBT instruction.
 - Is not performing register writeback.
 - Is not using R15 as a source or destination register.

For these stage 2 aborts, ISV is UNKNOWN if the exception was generated in Debug state in memory access mode, and otherwise indicates whether ISS[23:14] hold a valid syndrome.

For faults reported in ESR_EL1 or ESR_EL3, ISV is 1 when FEAT_LS64 is implemented and a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault. ISV is 0 for all other faults reported in ESR_EL1 or ESR_EL3.

When FEAT_RAS is implemented, ISV is 0 for any synchronous External abort.

For ISS reporting, a stage 2 abort on a stage 1 translation table walk does not return a valid instruction syndrome, and therefore ISV is 0 for these aborts.

When FEAT_RAS is not implemented, it is IMPLEMENTATION DEFINED whether ISV is set to 1 or 0 on a synchronous External abort on a stage 2 translation table walk.

When FEAT_MTE2 is implemented, for a synchronous Tag Check Fault abort taken to ELx, ESR_ELx.FNV is 0 and FAR_ELx is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SAS, bits [23:22]

When ISV == 1:

Syndrome Access Size. Indicates the size of the access attempted by the faulting operation.

| SAS | Meaning |
|------|------------|
| 0b00 | Byte |
| 0b01 | Halfword |
| 0b10 | Word |
| 0b11 | Doubleword |

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0b11.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSE, bit [21]**When ISV == 1:**

Syndrome Sign Extend. For a byte, halfword, or word load operation, indicates whether the data item must be sign extended.

| SSE | Meaning |
|-----|----------------------------------|
| 0b0 | Sign-extension not required. |
| 0b1 | Data item must be sign-extended. |

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

For all other operations, this field is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SRT, bits [20:16]**When ISV == 1:**

Syndrome Register Transfer. The register number of the Wt/Xt/Rt operand of the faulting instruction.

If the exception was taken from an Exception level that is using AArch32, then this is the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states'.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SF, bit [15]**When ISV == 1:**

Width of the register accessed by the instruction is Sixty-Four.

| SF | Meaning |
|-----|--|
| 0b0 | Instruction loads/stores a 32-bit wide register. |
| 0b1 | Instruction loads/stores a 64-bit wide register. |

Note

This field specifies the register width identified by the instruction, not the Execution state.

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 1.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AR, bit [14]

When ISV == 1:

Acquire/Release.

| AR | Meaning |
|-----|---|
| 0b0 | Instruction did not have acquire/release semantics. |
| 0b1 | Instruction did have acquire/release semantics. |

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VNCR, bit [13]

When FEAT_NV2 is implemented:

Indicates that the fault came from use of [VNCR_EL2](#) register by EL1 code.

| VNCR | Meaning |
|------|--|
| 0b0 | The fault was not generated by the use of VNCR_EL2 , by an MRS or MSR instruction executed at EL1. |
| 0b1 | The fault was generated by the use of VNCR_EL2 , by an MRS or MSR instruction executed at EL1. |

This field is 0 in ESR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SET, bits [12:11]

When FEAT_RAS is implemented and FEAT_LS64 is not implemented:

Synchronous Error Type. When DFSC is 0b010000, describes the PE error state after taking the Data Abort exception.

| SET | Meaning |
|------|--------------------------|
| 0b00 | Recoverable state (UER). |
| 0b10 | Uncontainable (UC). |
| 0b11 | Restartable state (UEO). |

All other values are reserved.

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in a PE state that is not recoverable.

This field is valid only if the DFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_LS64 is implemented:

Load/Store Type. Used when an LD64B, ST64B, ST64BV, or ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

| LST | Meaning |
|------|---|
| 0b01 | An ST64BV instruction generated the Data Abort. |
| 0b10 | An LD64B or ST64B instruction generated the Data Abort. |
| 0b11 | An ST64BV0 instruction generated the Data Abort. |

All other values are reserved.

This field is valid only if the DFSC code is 0b110101. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|---|
| 0b0 | FAR is valid. |
| 0b1 | FAR is not valid, and holds an UNKNOWN value. |

This field is valid only if the DFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CM, bit [8]

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:

| CM | Meaning |
|-----|--|
| 0b0 | The Data Abort was not generated by the execution of one of the System instructions identified in the description of value 1. |
| 0b1 | The Data Abort was generated by either the execution of a cache maintenance instruction or by a synchronous fault on the execution of an address translation instruction. The DC ZVA , DC GVA , and DC GZVA instructions are not classified as cache maintenance instructions, and therefore their execution cannot cause this field to be set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

| S1PTW | Meaning |
|-------|---|
| 0b0 | Fault not on a stage 2 translation for a stage 1 translation table walk. |
| 0b1 | Fault on the stage 2 translation of an access for a stage 1 translation table walk. |

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

| WnR | Meaning |
|-----|--|
| 0b0 | Abort caused by an instruction reading from a memory location. |
| 0b1 | Abort caused by an instruction writing to a memory location. |

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is UNKNOWN for:

- An External abort on an Atomic access.
- A fault reported using a DFSC value of 0b110101 or 0b110001, indicating an unsupported Exclusive or atomic access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code.

| DFSC | Meaning | Applies when |
|----------|---|--|
| 0b000000 | Address size fault, level 0 of translation or translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000100 | Translation fault, level 0. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001000 | Access flag fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001100 | Permission fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010000 | Synchronous External abort, not on translation table walk or hardware update of translation table. | |
| 0b010001 | Synchronous Tag Check Fault. | When FEAT_MTE2 is implemented |
| 0b010011 | Synchronous External abort on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented |
| 0b010100 | Synchronous External abort on translation table walk or hardware update of translation table, level 0. | |
| 0b010101 | Synchronous External abort on translation table walk or hardware update of translation table, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk or hardware update of translation table, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk or hardware update of translation table, level 3. | |
| 0b011000 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b011011 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented and FEAT_RAS is not implemented |
| 0b011100 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0. | When FEAT_RAS is not implemented |
| 0b011101 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1. | When FEAT_RAS is not implemented |

| | | |
|----------|--|---|
| 0b011110 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2. | When FEAT_RAS is not implemented |
| 0b011111 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3. | When FEAT_RAS is not implemented |
| 0b100001 | Alignment fault. | |
| 0b100011 | Granule Protection Fault on translation table walk or hardware update of translation table, level -1. | When FEAT_RME is implemented and FEAT_LPA2 is implemented |
| 0b100100 | Granule Protection Fault on translation table walk or hardware update of translation table, level 0. | When FEAT_RME is implemented |
| 0b100101 | Granule Protection Fault on translation table walk or hardware update of translation table, level 1. | When FEAT_RME is implemented |
| 0b100110 | Granule Protection Fault on translation table walk or hardware update of translation table, level 2. | When FEAT_RME is implemented |
| 0b100111 | Granule Protection Fault on translation table walk or hardware update of translation table, level 3. | When FEAT_RME is implemented |
| 0b101000 | Granule Protection Fault, not on translation table walk or hardware update of translation table. | When FEAT_RME is implemented |
| 0b101001 | Address size fault, level -1. | When FEAT_LPA2 is implemented |
| 0b101011 | Translation fault, level -1. | When FEAT_LPA2 is implemented |
| 0b110000 | TLB conflict abort. | |
| 0b110001 | Unsupported atomic hardware update fault. | When FEAT_HAFDBS is implemented |
| 0b110100 | IMPLEMENTATION DEFINED fault (Lockdown). | |
| 0b110101 | IMPLEMENTATION DEFINED fault (Unsupported Exclusive or Atomic access). | |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults'.

Note

Because Access flag faults and Permission faults can result only from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from a trapped floating-point exception

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|----|----|----|----|----|----|------|----|----|----|----|----|--------|-----|------|-----|-----|-----|-----|-----|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | TFV | | | | | | | RES0 | | | | | | VECITR | IDF | RES0 | IXF | UFF | OFF | DZF | IOF | | | |

Bit [24]

Reserved, RES0.

TFV, bit [23]

Trapped Fault Valid bit. Indicates whether the IDF, IXF, UFF, OFF, DZF, and IOF bits hold valid information about trapped floating-point exceptions.

| TFV | Meaning |
|-----|---|
| 0b0 | The IDF, IXF, UFF, OFF, DZF, and IOF bits do not hold valid information about trapped floating-point exceptions and are UNKNOWN. |
| 0b1 | One or more floating-point exceptions occurred during an operation performed while executing the reported instruction. The IDF, IXF, UFF, OFF, DZF, and IOF bits indicate trapped floating-point exceptions that occurred. For more information, see 'Floating-point exceptions and exception traps'. |

It is IMPLEMENTATION DEFINED whether this field is set to 0 on an exception generated by a trapped floating-point exception from an instruction that is performing floating-point operations on more than one lane of a vector.

Note

This is not a requirement. Implementations can set this field to 1 on a trapped floating-point exception from an instruction and return valid information in the {IDF, IXF, UFF, OFF, DZF, IOF} fields.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [22:11]

Reserved, RES0.

VECITR, bits [10:8]

For a trapped floating-point exception from an instruction executed in AArch32 state this field is RES1.

For a trapped floating-point exception from an instruction executed in AArch64 state this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| IDF | Meaning |
|-----|--|
| 0b0 | Input denormal floating-point exception has not occurred. |
| 0b1 | Input denormal floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| IXF | Meaning |
|-----|---|
| 0b0 | Inexact floating-point exception has not occurred. |
| 0b1 | Inexact floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFF, bit [3]

Underflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| UFF | Meaning |
|-----|---|
| 0b0 | Underflow floating-point exception has not occurred. |
| 0b1 | Underflow floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFF, bit [2]

Overflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| OFF | Meaning |
|-----|--|
| 0b0 | Overflow floating-point exception has not occurred. |
| 0b1 | Overflow floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZF, bit [1]

Divide by Zero floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| DZF | Meaning |
|-----|--|
| 0b0 | Divide by Zero floating-point exception has not occurred. |
| 0b1 | Divide by Zero floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOF, bit [0]

Invalid Operation floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| IOF | Meaning |
|-----|---|
| 0b0 | Invalid Operation floating-point exception has not occurred. |
| 0b1 | Invalid Operation floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

In an implementation that supports the trapping of floating-point exceptions:

- From an Exception level using AArch64, the [FPCR](#).{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.
- From an Exception level using AArch32, the [FPSCR](#).{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.

ISS encoding for an SError interrupt

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|------|----|----|----|----|----|----|----|----|----|------|----|-----|---|----|---|------|---|---|------|---|---|---|--|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| IDS | | RES0 | | | | | | | | | | IESB | | AET | | EA | | RES0 | | | DFSC | | | | |

IDS, bit [24]

IMPLEMENTATION DEFINED syndrome.

| IDS | Meaning |
|---|---|
| 0b0 | Bits [23:0] of the ISS field holds the fields described in this encoding. |
| Note If FEAT_RAS is not implemented, bits [23:0] of the ISS field are RES0. | |
| 0b1 | Bits [23:0] of the ISS field holds IMPLEMENTATION DEFINED syndrome information that can be used to provide additional information about the SError interrupt. |

Note

This field was previously called ISV.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:14]

Reserved, RES0.

IESB, bit [13]

When **FEAT_IESB** is implemented:

Implicit error synchronization event.

| IESB | Meaning |
|-------------|--|
| 0b0 | The SError interrupt was either not synchronized by the implicit error synchronization event or not taken immediately. |
| 0b1 | The SError interrupt was synchronized by the implicit error synchronization event and taken immediately. |

This field is valid only if the DFSC code is 0b010001. It is RES0 for all other errors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AET, bits [12:10]

When **FEAT_RAS** is implemented:

Asynchronous Error Type.

When DFSC is 0b010001, describes the PE error state after taking the SError interrupt exception.

| AET | Meaning |
|------------|----------------------------|
| 0b000 | Uncontainable (UC). |
| 0b001 | Unrecoverable state (UEU). |
| 0b010 | Restartable state (UEO). |
| 0b011 | Recoverable state (UER). |
| 0b110 | Corrected (CE). |

All other values are reserved.

If multiple errors are taken as a single SError interrupt exception, the overall PE error state is reported.

Note

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

This field is valid only if the DFSC code is 0b010001. It is RES0 for all other errors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EA, bit [9]**When FEAT_RAS is implemented:**

External abort type. When DFSC is 0b010001, provides an IMPLEMENTATION DEFINED classification of External aborts.

This field is valid only if the DFSC code is 0b010001. It is RES0 for all other errors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [8:6]

Reserved, RES0.

DFSC, bits [5:0]**When FEAT_RAS is implemented:**

Data Fault Status Code.

| DFSC | Meaning |
|----------|--------------------------------|
| 0b000000 | Uncategorized error. |
| 0b010001 | Asynchronous SError interrupt. |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ISS encoding for an exception from a Breakpoint or Vector Catch debug exception

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|------|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | IFSC | | | | | |

Bits [24:6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code.

| IFSC | Meaning |
|----------|------------------|
| 0b100010 | Debug exception. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions:

- For exceptions from AArch64, see 'Breakpoint exceptions'.
- For exceptions from AArch32, see 'Breakpoint exceptions' and 'Vector Catch exceptions'.

ISS encoding for an exception from a Software Step exception

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|----|---|------|---|---|---|--|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| ISV | | RES0 | | | | | | | | | | | | | | | | | EX | | IFSC | | | | |

ISV, bit [24]

Instruction syndrome valid. Indicates whether the EX bit, ISS[6], is valid, as follows:

| ISV | Meaning |
|-----|------------------|
| 0b0 | EX bit is RES0. |
| 0b1 | EX bit is valid. |

See the EX bit description for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:7]

Reserved, RES0.

EX, bit [6]

Exclusive operation. If the ISV bit is set to 1, this bit indicates whether a Load-Exclusive instruction was stepped.

| EX | Meaning |
|-----|---|
| 0b0 | An instruction other than a Load-Exclusive instruction was stepped. |
| 0b1 | A Load-Exclusive instruction was stepped. |

If the ISV bit is set to 0, this bit is RES0, indicating no syndrome data is available.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IFSC, bits [5:0]

Instruction Fault Status Code.

| IFSC | Meaning |
|----------|------------------|
| 0b100010 | Debug exception. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Software Step exceptions'.

ISS encoding for an exception from a Watchpoint exception

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----------|----|------|----|----|---|-----------|---|---|------|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | RES0VNCR | | RES0 | | | | CMRES0WnR | | | DFSC | | | | | |

Bits [24:15]

Reserved, RES0.

Bit [14]

Reserved, RES0.

VNCR, bit [13]

When FEAT_NV2 is implemented:

Indicates that the watchpoint came from use of [VNCR_EL2](#) register by EL1 code.

| VNCR | Meaning |
|------|--|
| 0b0 | The watchpoint was not generated by the use of VNCR_EL2 by EL1 code. |
| 0b1 | The watchpoint was generated by the use of VNCR_EL2 by EL1 code. |

This field is 0 in ESR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [12:9]

Reserved, RES0.

CM, bit [8]

Cache maintenance. Indicates whether the Watchpoint exception came from a cache maintenance or address translation instruction:

| CM | Meaning |
|-----|---|
| 0b0 | The Watchpoint exception was not generated by the execution of one of the System instructions identified in the description of value 1. |
| 0b1 | The Watchpoint exception was generated by either the execution of a cache maintenance instruction or by a synchronous Watchpoint exception on the execution of an address translation instruction. The DC ZVA , DC GVA , and DC GZVA instructions are not classified as a cache maintenance instructions, and therefore their execution cannot cause this field to be set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

WnR, bit [6]

Write not Read. Indicates whether the Watchpoint exception was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

| WnR | Meaning |
|-----|---|
| 0b0 | Watchpoint exception caused by an instruction reading from a memory location. |
| 0b1 | Watchpoint exception caused by an instruction writing to a memory location. |

For Watchpoint exceptions on cache maintenance and address translation instructions, this bit always returns a value of 1.

For Watchpoint exceptions from an atomic instruction, this field is set to 0 if a read of the location would have generated the Watchpoint exception, otherwise it is set to 1.

If multiple watchpoints match on the same access, it is UNPREDICTABLE which watchpoint generates the Watchpoint exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code.

| DFSC | Meaning |
|----------|------------------|
| 0b100010 | Debug exception. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Watchpoint exceptions'.

ISS encoding for an exception from execution of a Breakpoint instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | Comment | | | | | | | | | | | | | | | |

Bits [24:16]

Reserved, RES0.

Comment, bits [15:0]

Set to the instruction comment field value, zero extended as necessary.

For the AArch32 BKPT instructions, the comment field is described as the immediate field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Breakpoint instruction exceptions'.

ISS encoding for an exception from an ERET, ERETAA, or ERETAB instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|------|---|-------|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | ERET | | ERETA |

This EC value applies when FEAT_FGT is implemented, or when [HCR_EL2.NV](#) is 1.

Bits [24:2]

Reserved, RES0.

ERET, bit [1]

Indicates whether an ERET or ERETA* instruction was trapped to EL2.

| ERET | Meaning |
|------|--|
| 0b0 | ERET instruction trapped to EL2. |
| 0b1 | ERETAA or ERETAB instruction trapped to EL2. |

If this bit is 0, the ERETA field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ERETA, bit [0]

Indicates whether an ERETAA or ERETAB instruction was trapped to EL2.

| ERETA | Meaning |
|-------|------------------------------------|
| 0b0 | ERETAA instruction trapped to EL2. |
| 0b1 | ERETAB instruction trapped to EL2. |

When the ERET field is 0, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see [HCR_EL2.NV](#).

If FEAT_FGT is implemented, [HFGITR_EL2.ERET](#) controls fine-grained trap exceptions from ERET, ERETAA and ERETAB execution.

ISS encoding for an exception from a TSTART instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|------|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | Rd | | | RES0 | | | | | | |

Bits [24:10]

Reserved, RES0.

Rd, bits [9:5]

The Rd value from the issued instruction, the general purpose register used for the destination.

Bits [4:0]

Reserved, RES0.

ISS encoding for an exception from Branch Target Identification instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|--------|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | BTTYPE | |

Bits [24:2]

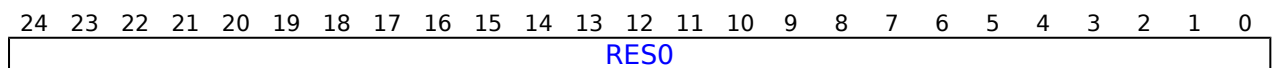
Reserved, RES0.

BTYPE, bits [1:0]

This field is set to the PSTATE.BTYPE value that generated the Branch Target Exception.

For more information about generating these exceptions, see 'The AArch64 application level programmers model'.

ISS encoding for an exception from a Pointer Authentication instruction when HCR_EL2.API == 0 || SCR_EL3.API == 0

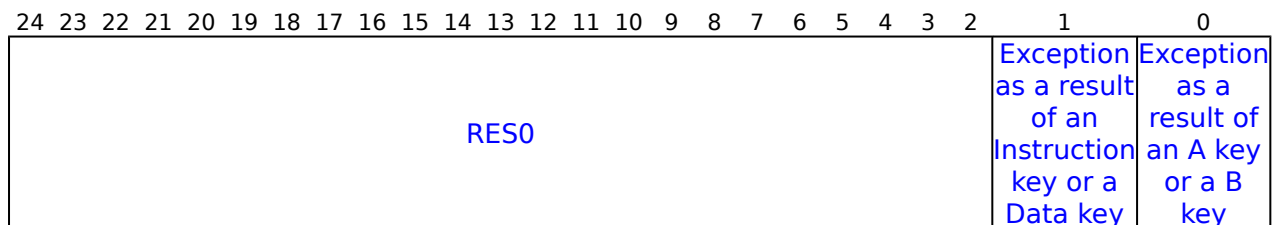
**Bits [24:0]**

Reserved, RES0.

For more information about generating these exceptions, see:

- [HCR_EL2.API](#), for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL2.
- [SCR_EL3.API](#), for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL3.

ISS encoding for an exception from a Pointer Authentication instruction authentication failure

**Bits [24:2]**

Reserved, RES0.

Bit [1]

This field indicates whether the exception is as a result of an Instruction key or a Data key.

| Meaning | |
|---------|------------------|
| 0b0 | Instruction Key. |
| 0b1 | Data Key. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

This field indicates whether the exception is as a result of an A key or a B key.

| | Meaning |
|-----|---------|
| 0b0 | A key. |
| 0b1 | B key. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following instructions generate an exception when the Pointer Authentication Code (PAC) is incorrect:

- AUTIASP, AUTIAZ, AUTIA1716.
- AUTIBSP, AUTIBZ, AUTIB1716.
- AUTIA, AUTDA, AUTIB, AUTDB.
- AUTIZA, AUTIZB, AUTDZA, AUTDZB.

It is IMPLEMENTATION DEFINED whether the following instructions generate an exception directly from the authorization failure, rather than changing the address in a way that will generate a Translation fault when the address is accessed:

- RETAA, RETAB.
- BRAA, BRAB, BLRAA, BLRAB.
- BRAAZ, BRABZ, BLRAAZ, BLRABZ.
- ERETA, ERETAB.
- LDRAA, LDRAB, whether the authenticated address is written back to the base register or not.

Accessing ESR_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic ESR_EL2 or ESR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ESR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return ESR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return ESR_EL2;
elsif PSTATE.EL == EL3 then
    return ESR_EL2;

```

MSR ESR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        ESR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    ESR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    ESR_EL2 = X[t];

```

MRS <Xt>, ESR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ESR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x138];
    else
        return ESR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return ESR_EL2;
    else
        return ESR_EL1;
elsif PSTATE.EL == EL3 then
    return ESR_EL1;

```

MSR ESR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ESR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x138] = X[t];
    else
        ESR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        ESR_EL2 = X[t];
    else
        ESR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    ESR_EL1 = X[t];

```


ESR_EL3, Exception Syndrome Register (EL3)

The ESR_EL3 characteristics are:

Purpose

Holds syndrome information for an exception taken to EL3.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to ESR_EL3 are UNDEFINED.

Attributes

ESR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ISS2 | | | |
| EC | | | | | | IL | | ISS | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

ESR_EL3 is made UNKNOWN as a result of an exception return from EL3.

When an UNPREDICTABLE instruction is treated as UNDEFINED, and the exception is taken to EL3, the value of ESR_EL3 is UNKNOWN. The value written to ESR_EL3 must be consistent with a value that could be created as a result of an exception from the same Exception level that generated the exception as a result of a situation that is not UNPREDICTABLE at that Exception level, in order to avoid the possibility of a privilege violation.

Bits [63:37]

Reserved, RES0.

ISS2, bits [36:32]

When FEAT_LS64 is implemented:

If a memory access generated by an ST64BV or ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field holds register specifier, Xs.

For any other Data Abort, this field is RES0.

Otherwise:

Reserved, RES0.

EC, bits [31:26]

Exception Class. Indicates the reason for the exception that this register holds information about.

For each EC value, the table references a subsection that gives information about:

- The cause of the exception, for example the configuration required to enable the trap.
- The encoding of the associated ISS.

Possible values of the EC field are:

| EC | Meaning | ISS | Applies when |
|----------|--|---|---------------------------|
| 0b000000 | Unknown reason. | ISS encoding for exceptions with an unknown reason | |
| 0b000001 | Trapped WF* instruction execution. Conditional WF* instructions that fail their condition code check do not cause an exception. | ISS encoding for an exception from a WF* instruction | |
| 0b000011 | Trapped MCR or MRC access with (coproc==0b1111) that is not reported using EC 0b000000. | ISS encoding for an exception from an MCR or MRC access | When AArch32 is supported |
| 0b000100 | Trapped MCRR or MRRC access with (coproc==0b1111) that is not reported using EC 0b000000. | ISS encoding for an exception from an MCRR or MRRC access | When AArch32 is supported |
| 0b000101 | Trapped MCR or MRC access with (coproc==0b1110). | ISS encoding for an exception from an MCR or MRC access | When AArch32 is supported |
| 0b000110 | Trapped LDC or STC access. The only architected uses of these instruction are: <ul style="list-style-type: none"> An STC to write data to memory from DBGDTRRXint. An LDC to read data from memory to DBGDTRTXint. | ISS encoding for an exception from an LDC or STC instruction | When AArch32 is supported |
| 0b000111 | Access to SME, SVE, Advanced SIMD or floating-point functionality trapped by CPACR_EL1 .FPEN, CPTR_EL2 .FPEN, CPTR_EL2 .TFP, or CPTR_EL3 .TFP control. Excludes exceptions resulting from CPACR_EL1 when the value of HCR_EL2 .TGE is 1, or because SVE or Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000 as described in 'The EC used to report an exception routed to EL2 because HCR_EL2.TGE is 1'. | ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from the FPEN and TFP traps | |
| 0b001001 | Trapped use of a Pointer authentication | ISS encoding for an exception from a Pointer | When FEAT_PAuth |

| | | | |
|----------|--|---|-------------------------------|
| | instruction because HCR_EL2.API == 0 SCR_EL3.API == 0 . | Authentication instruction when HCR_EL2.API == 0 SCR_EL3.API == 0 | is implemented |
| 0b001010 | Trapped execution of an LD64B, ST64B, ST64BV, or ST64BV0 instruction. | ISS encoding for an exception from an LD64B or ST64B* instruction | When FEAT_LS64 is implemented |
| 0b001100 | Trapped MRRC access with (coproc==0b1110). | ISS encoding for an exception from an MCRR or MRRC access | When AArch32 is supported |
| 0b001101 | Branch Target Exception. | ISS encoding for an exception from Branch Target Identification instruction | When FEAT_BTI is implemented |
| 0b001110 | Illegal Execution state. | ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault | |
| 0b010011 | SMC instruction execution in AArch32 state, when SMC is not disabled. | ISS encoding for an exception from SMC instruction execution in AArch32 state | When AArch32 is supported |
| 0b010101 | SVC instruction execution in AArch64 state. | ISS encoding for an exception from HVC or SVC instruction execution | When AArch64 is supported |
| 0b010110 | HVC instruction execution in AArch64 state, when HVC is not disabled. | ISS encoding for an exception from HVC or SVC instruction execution | When AArch64 is supported |
| 0b010111 | SMC instruction execution in AArch64 state, when SMC is not disabled. | ISS encoding for an exception from SMC instruction execution in AArch64 state | When AArch64 is supported |
| 0b011000 | Trapped MSR, MRS or System instruction execution in AArch64 state, that is not reported using EC 0b000000, 0b000001 or 0b000111. This includes all instructions that cause exceptions that are part of the encoding space defined in 'System instruction class encoding overview', except for those exceptions reported using EC values 0b000000, 0b000001, or 0b000111. | ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state | When AArch64 is supported |
| 0b011001 | Access to SVE functionality trapped as a result of CPACR_EL1.ZEN , | ISS encoding for an exception from an access to SVE functionality | When FEAT_SVE is implemented |

| | | | |
|----------|---|--|--------------------------------------|
| 0b011011 | <p>CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ, that is not reported using EC 0b000000. Exception from an access to a TSTART instruction at EL0 when SCTLR_EL1.TME0 == 0, EL0 when SCTLR_EL2.TME0 == 0, at EL1 when SCTLR_EL1.TME == 0, at EL2 when SCTLR_EL2.TME == 0 or at EL3 when SCTLR_EL3.TME == 0.</p> | <p>resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ ISS encoding for an exception from a TSTART instruction</p> | <p>When FEAT_TME is implemented</p> |
| 0b011100 | <p>Exception from a Pointer Authentication instruction authentication failure</p> | <p>ISS encoding for an exception from a Pointer Authentication instruction authentication failure</p> | <p>When FEAT_FPAC is implemented</p> |
| 0b011101 | <p>Access to SME functionality trapped as a result of CPACR_EL1.SMEN, CPTR_EL2.SMEN, CPTR_EL2.TSM, CPTR_EL3.ESM, or an attempted execution of an instruction that is illegal because of the value of PSTATE.SM or PSTATE.ZA, that is not reported using EC 0b000000.</p> | <p>ISS encoding for an exception due to SME functionality</p> | <p>When FEAT_SME is implemented</p> |
| 0b011110 | <p>Exception from a Granule Protection Check</p> | <p>ISS encoding for an exception from a Granule Protection Check</p> | <p>When FEAT_RME is implemented</p> |
| 0b011111 | <p>IMPLEMENTATION DEFINED exception to EL3.</p> | <p>ISS encoding for an IMPLEMENTATION DEFINED exception to EL3</p> | |
| 0b100000 | <p>Instruction Abort from a lower Exception level. Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions.</p> | <p>ISS encoding for an exception from an Instruction Abort</p> | |
| 0b100001 | <p>Instruction Abort taken without a change in Exception level.</p> | <p>ISS encoding for an exception from an Instruction Abort</p> | |

| | | | |
|----------|---|--|-------------------------------|
| 0b100010 | Used for MMU faults generated by instruction accesses and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions. PC alignment fault exception. | ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault | |
| 0b100100 | Data Abort from a lower Exception level. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions. | ISS encoding for an exception from a Data Abort | |
| 0b100101 | Data Abort taken without a change in Exception level. Used for MMU faults generated by data accesses, alignment faults other than those caused by Stack Pointer misalignment, and synchronous External aborts, including synchronous parity or ECC errors. Not used for debug-related exceptions. | ISS encoding for an exception from a Data Abort | |
| 0b100110 | SP alignment fault exception. | ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault | |
| 0b100111 | Memory Operation Exception. | ISS encoding for an exception from the Memory Copy and Memory Set instructions | When FEAT_MOPS is implemented |
| 0b101100 | Trapped floating-point exception taken from AArch64 state. This EC value is valid if the implementation supports trapping of floating-point | ISS encoding for an exception from a trapped floating-point exception | When AArch64 is supported |

| | | | |
|----------|---|--|---------------------------------|
| | exceptions, otherwise it is reserved. Whether a floating-point implementation supports trapping of floating-point exceptions is IMPLEMENTATION DEFINED. | | |
| 0b101111 | SError interrupt. | ISS encoding for an SError interrupt | |
| 0b111100 | BRK instruction execution in AArch64 state. This is reported in ESR_EL3 only if a BRK instruction is executed in EL3. This is the only debug exception that can be taken to EL3 when EL3 is using AArch64. | ISS encoding for an exception from execution of a Breakpoint instruction | When AArch64 is supported |

All other EC values are reserved by Arm, and:

- Unused values in the range 0b000000 - 0b101100 (0x00 - 0x2C) are reserved for future use for synchronous exceptions.
- Unused values in the range 0b101101 - 0b111111 (0x2D - 0x3F) are reserved for future use, and might be used for synchronous or asynchronous exceptions.

The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [25]

Instruction Length for synchronous exceptions. Possible values of this bit are:

| IL | Meaning |
|-----|---|
| 0b0 | 16-bit instruction trapped. |
| 0b1 | 32-bit instruction trapped. This value is also used when the exception is one of the following: <ul style="list-style-type: none"> • An SError interrupt. • An Instruction Abort exception. • A PC alignment fault exception. • An SP alignment fault exception. • A Data Abort exception for which the value of the ISV bit is 0. • An Illegal Execution state exception. • Any debug exception except for Breakpoint instruction exceptions. • An exception reported using EC value 0b000000. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [24:0]

Instruction Specific Syndrome. Architecturally, this field can be defined independently for each defined Exception class. However, in practice, some ISS encodings are used for more than one Exception class.

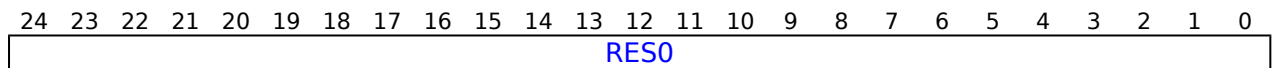
Typically, an ISS encoding has a number of subfields. When an ISS subfield holds a register number, the value returned in that field is the AArch64 view of the register number.

For an exception taken from AArch32 state, see 'Mapping of the general-purpose registers between the Execution states'.

If the AArch32 register descriptor is 0b1111, then:

- If the instruction that generated the exception was not UNPREDICTABLE, the field takes the value 0b11111.
- If the instruction that generated the exception was UNPREDICTABLE, the field takes an UNKNOWN value that must be either:
 - The AArch64 view of the register number of a register that might have been used at the Exception level from which the exception was taken.
 - The value 0b11111.

ISS encoding for exceptions with an unknown reason



Bits [24:0]

Reserved, RES0.

When an exception is reported using this EC code the IL field is set to 1.

This EC code is used for all exceptions that are not covered by any other EC value. This includes exceptions that are generated in the following situations:

- The attempted execution of an instruction bit pattern that has no allocated instruction or that is not accessible at the current Exception level and Security state, including:
 - A read access using a System register pattern that is not allocated for reads or that does not permit reads at the current Exception level and Security state.
 - A write access using a System register pattern that is not allocated for writes or that does not permit writes at the current Exception level and Security state.
 - Instruction encodings that are unallocated.
 - Instruction encodings for instructions or System registers that are not implemented in the implementation.
- In Debug state, the attempted execution of an instruction bit pattern that is not accessible in Debug state.
- In Non-debug state, the attempted execution of an instruction bit pattern that is not accessible in Non-debug state.
- In AArch32 state, attempted execution of a short vector floating-point instruction.
- In an implementation that does not include Advanced SIMD and floating-point functionality, an attempted access to Advanced SIMD or floating-point functionality under conditions where that access would be permitted if that functionality was present. This includes the attempted execution of an Advanced SIMD or floating-point instruction, and attempted accesses to Advanced SIMD and floating-point System registers.
- An exception generated because of the value of one of the [SCTLR_EL1](#).{ITD, SED, CP15BEN} control bits.
- Attempted execution of:
 - An HVC instruction when disabled by [HCR_EL2](#).HCD or [SCR_EL3](#).HCE.
 - An SMC instruction when disabled by [SCR_EL3](#).SMD.
 - An HLT instruction when disabled by [EDSCR](#).HDE.
- Attempted execution of an MSR or MRS instruction to access [SP_EL0](#) when the value of [SPSel](#).SP is 0.
- Attempted execution of an MSR or MRS instruction using a [_EL12](#) register name when [HCR_EL2](#).E2H == 0.
- Attempted execution, in Debug state, of:
 - A DCPS1 instruction when the value of [HCR_EL2](#).TGE is 1 and EL2 is disabled or not implemented in the current Security state.
 - A DCPS2 instruction from EL1 or EL0 when EL2 is disabled or not implemented in the current Security state.
 - A DCPS3 instruction when the value of [EDSCR](#).SDD is 1, or when EL3 is not implemented.
- When EL3 is using AArch64, attempted execution from Secure EL1 of an SRS instruction using R13_mon. See 'Traps to EL3 of Secure monitor functionality from Secure EL1 using AArch32'.
- In Debug state when the value of [EDSCR](#).SDD is 1, the attempted execution at EL2, EL1, or EL0 of an instruction that is configured to trap to EL3.
- In AArch32 state, the attempted execution of an MRS (banked register) or an MSR (banked register) instruction to [SPSR_mon](#), [SP_mon](#), or [LR_mon](#).

- An exception that is taken to EL2 because the value of [HCR_EL2.TGE](#) is 1 that, if the value of [HCR_EL2.TGE](#) was 0 would have been reported with an [ESR_ELx.EC](#) value of 0b000111.
- In Non-transactional state, attempted execution of a TCOMMIT instruction.

ISS encoding for an exception from a WF* instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|----|----|----|----|----|----|----|---|---|---|------|---|----|----|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | RES0 | | | | | | | | | | RN | | | | RES0 | | RV | TI | | |

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:10]

Reserved, RES0.

RN, bits [9:5]**When FEAT_WFxT is implemented:**

Register Number. Indicates the register number supplied for a WFET or WFIT instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [4:3]

Reserved, RES0.

RV, bit [2]**When FEAT_WFxT is implemented:**

Register field Valid.

If TI[1] == 1, then this field indicates whether RN holds a valid register number for the register argument to the trapped WFET or WFIT instruction.

| RV | Meaning |
|-----------|-------------------------|
| 0b0 | Register field invalid. |
| 0b1 | Register field valid. |

If TI[1] == 0, then this field is RES0.

This field is set to 1 on a trap on WFET or WFIT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TI, bits [1:0]

Trapped instruction. Possible values of this bit are:

| TI | Meaning | Applies when |
|-----------|----------------|-------------------------------|
| 0b00 | WFI trapped. | |
| 0b01 | WFE trapped. | |
| 0b10 | WFIT trapped. | When FEAT_WFxT is implemented |
| 0b11 | WFET trapped. | When FEAT_WFxT is implemented |

When FEAT_WFxT is implemented, this is a two bit field as shown. Otherwise, bit[1] is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following fields describe configuration settings for generating this exception:

- [SCTLR_EL1](#).{nTWE, nTWI}.
- [HCR_EL2](#).{TWE, TWI}.
- [SCR_EL3](#).{TWE, TWI}.

ISS encoding for an exception from an MCR or MRC access

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|------|----|----|-----|----|----|----|---|---|-----|---|---|-----------|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | Opc2 | | | Opc1 | | | CRn | | | Rt | | | CRm | | | Direction | | | | |

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc2, bits [19:17]

The Opc2 value from the issued instruction.

For a trapped VMRS access, holds the value 0b000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [16:14]

The Opc1 value from the issued instruction.

For a trapped VMRS access, holds the value 0b111.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

For a trapped VMRS access, holds the reg field from the VMRS instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

If the Rt value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b1111.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
 - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
 - The value 0b1111.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

For a trapped VMRS access, holds the value 0b0000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|---|
| 0b0 | Write to System register space. MCR instruction. |
| 0b1 | Read from System register space. MRC or VMRS instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000011:

- [CNTKCTL_EL1](#).{ELOPTEN, EL0VTEN, ELOPCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [PMUSERENR_EL0](#).{ER, CR, SW, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [AMUSERENR_EL0](#).EN, for accesses to Activity Monitors registers from EL0 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [HCR_EL2](#).{TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).TTLB, for execution of TLB maintenance instructions at EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).{TSW, TPC, TPU} for execution of cache maintenance instructions at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).TACR, for accesses to the Auxiliary Control Register at EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).TIDCP, for accesses to lockdown, DMA, and TCM operations at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HCR_EL2](#).{TID1, TID2, TID3}, for accesses to ID registers at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [CPTR_EL2](#).TCPAC, for accesses to [CPACR_EL1](#) or [CPACR](#) using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [HSTR_EL2](#).T<n>, for accesses to System registers using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [CNTHCTL_EL2](#).EL1PCEN, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [MDCR_EL2](#).{TPM, TPMCR}, for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [CPTR_EL2](#).TAM, for accesses to Activity Monitors registers from EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL2.
- [CPTR_EL3](#).TCPAC, for accesses to [CPACR](#) from EL1 and EL2, and accesses to [HCPTR](#) from EL2 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL3.
- [MDCR_EL3](#).TPM, for accesses to Performance Monitor registers from EL0, EL1 and EL2 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL3.
- [CPTR_EL3](#).TAM, for accesses to Activity Monitors registers from EL0, EL1 and EL2 using AArch32 state, MCR or MRC access (coproc == 0b1111) trapped to EL3.
- For information on other traps using EC value 0b000011, see 'Traps to EL3 of Secure monitor functionality from Secure EL1 using AArch32'.
- If FEAT_FGT is implemented, MCR or MRC access to some registers at EL0, trapped to EL2.

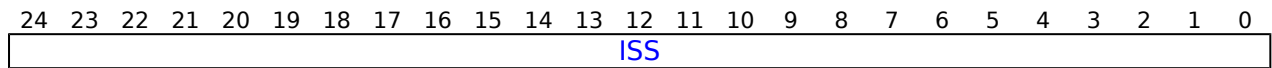
The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000101:

- [CPACR_EL1](#).TTA for accesses to trace registers, MCR or MRC access (coproc == 0b1110) trapped to EL1 or EL2.
- [MDCR_EL1](#).TDCC, for accesses to the Debug Communications Channel (DCC) registers at EL0 and EL1 using AArch32 state, MCR or MRC access (coproc == 0b1110) trapped to EL1 or EL2.
- If FEAT_FGT is implemented, [MDCR_EL2](#).TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and [MDCR_EL3](#).TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.
- [HCR_EL2](#).TID0, for accesses to the [JIDR](#) register in the ID group 0 at EL0 and EL1 using AArch32, MRC access (coproc == 0b1110) trapped to EL2.
- [CPTR_EL2](#).TTA, for accesses to trace registers using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- [MDCR_EL2](#).TDRA, for accesses to Debug ROM registers [DBGDRAR](#) and [DBGDSAR](#) using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- [MDCR_EL2](#).TDOSA, for accesses to powerdown debug registers, using AArch32 state, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- [MDCR_EL2](#).TDA, for accesses to other debug registers, using AArch32 state, MCR or MRC access (coproc == 0b1110) trapped to EL2.
- [CPTR_EL3](#).TTA, for accesses to trace registers using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL3.
- [MDCR_EL3](#).TDOSA, for accesses to powerdown debug registers using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL3.
- [MDCR_EL3](#).TDA, for accesses to other debug registers, using AArch32, MCR or MRC access (coproc == 0b1110) trapped to EL3.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b001000:

- [HCR_EL2.TID0](#), for accesses to the [FPSID](#) register in ID group 0 at EL1 using AArch32 state, VMRS access trapped to EL2.
- [HCR_EL2.TID3](#), for accesses to registers in ID group 3 including [MVFR0](#), [MVFR1](#) and [MVFR2](#), VMRS access trapped to EL2.

ISS encoding for an exception from an LD64B or ST64B* instruction

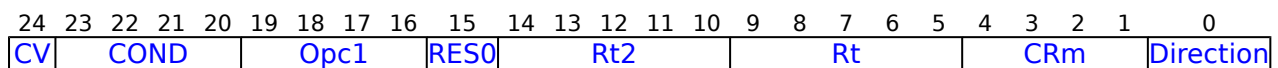


ISS, bits [24:0]

| ISS | Meaning |
|----------------------------------|-------------------------------------|
| 0b000000000000000000000000000000 | ST64BV instruction trapped. |
| 0b000000000000000000000000000001 | ST64BV0 instruction trapped. |
| 0b000000000000000000000000000010 | LD64B or ST64B instruction trapped. |

All other values are reserved.

ISS encoding for an exception from an MCRR or MRRC access



CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to 0b1110, the value for unconditional.
- With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [19:16]

The Opc1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

Rt2, bits [14:10]

The Rt2 value from the issued instruction, the second general-purpose register used for the transfer.

If the Rt2 value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt2 value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b1111.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
 - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
 - The value 0b1111.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the first general-purpose register used for the transfer.

If the Rt value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rt value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b1111.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
 - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.

- The value 0b11111.

See 'Mapping of the general-purpose registers between the Execution states'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|--|
| 0b0 | Write to System register space. MCRR instruction. |
| 0b1 | Read from System register space. MRRC instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b000100:

- [CNTKCTL_EL1](#).{ELOPTEN, EL0VTEN, ELOPCTEN, EL0VCTEN}, for accesses to the Generic Timer Registers from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [PMUSERENR_EL0](#).{CR, EN}, for accesses to Performance Monitor registers from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [AMUSERENR_EL0](#).{EN}, for accesses to Activity Monitors registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL1 or EL2.
- [HCR_EL2](#).{TRVM, TVM}, for accesses to virtual memory control registers from EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [HSTR_EL2](#).T<n>, for accesses to System registers using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [CNTHCTL_EL2](#).{EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [MDCR_EL2](#).{TPM, TPMCR}, for accesses to Performance Monitor registers from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [CPTR_EL2](#).TAM, for accesses to Activity Monitors registers AMEVCNTR0<n> and AMEVCNTR1<n> from EL0 and EL1 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL2.
- [MDCR_EL3](#).TPM, for accesses to Performance Monitor registers from EL0, EL1 and EL2 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL3.
- [CPTR_EL3](#).TAM, for accesses to Activity Monitors registers from EL0, EL1 and EL2 using AArch32 state, MCRR or MRRC access (coproc == 0b1111) trapped to EL3.
- If FEAT_FGT is implemented, [HDEGTR_EL2](#).PMCCNTR_EL0 for MRRC access and [HDEGWTR_EL2](#).PMCCNTR_EL0 for MCRR access to [PMCCNTR](#) at EL0, trapped to EL2.

The following fields describe configuration settings for generating exceptions that are reported using EC value 0b001100:

- [MDSCR_EL1](#).TDCC, for accesses to the Debug ROM registers [DBGDSAR](#) and [DBGDRAR](#) at EL0 using AArch32 state, MCRR or MRRC access (coproc == 0b1110) trapped to EL1 or EL2.
- [MDCR_EL2](#).TDRA, for accesses to Debug ROM registers [DBGDRAR](#) and [DBGDSAR](#) using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL2.
- [MDCR_EL3](#).TDA, for accesses to debug registers, using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL3.

- [CPACR_EL1](#).TTA for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL1 or EL2.
- [CPTR_EL2](#).TTA, for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL2.
- [CPTR_EL3](#).TTA, for accesses to trace registers using AArch32, MCRR or MRRC access (coproc == 0b1110) trapped to EL3.

Note

If the Armv8-A architecture is implemented with an ETMv4 implementation, MCRR and MRRC accesses to trace registers are UNDEFINED and the resulting exception is higher priority than an exception due to these traps.

ISS encoding for an exception from an LDC or STC instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|----|----|----|----|------|----|---|---|---|--------|---|---|---|----|-----------|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | imm8 | | | | | | | | RES0 | Rn | | | | Offset | | | | AM | Direction | |

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

imm8, bits [19:12]

The immediate value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

Rn, bits [9:5]

The Rn value from the issued instruction, the general-purpose register used for the transfer.

If the Rn value is not 0b1111, then the reported value gives the AArch64 view of the register. Otherwise, if the Rn value is 0b1111:

- If the instruction that generated the exception is not UNPREDICTABLE, then the register specifier takes the value 0b1111.
- If the instruction that generated the exception is UNPREDICTABLE, then the register specifier takes an UNKNOWN value, which is restricted to either:
 - The AArch64 view of one of the registers that could have been used in AArch32 state at the Exception level that the instruction was executed at.
 - The value 0b1111.

See 'Mapping of the general-purpose registers between the Execution states'.

This field is valid only when AM[2] is 0, indicating an immediate form of the LDC or STC instruction. When AM[2] is 1, indicating a literal form of the LDC or STC instruction, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Offset, bit [4]

Indicates whether the offset is added or subtracted:

| Offset | Meaning |
|--------|------------------|
| 0b0 | Subtract offset. |
| 0b1 | Add offset. |

This bit corresponds to the U bit in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AM, bits [3:1]

Addressing mode. The permitted values of this field are:

| AM | Meaning |
|-------|---|
| 0b000 | Immediate unindexed. |
| 0b001 | Immediate post-indexed. |
| 0b010 | Immediate offset. |
| 0b011 | Immediate pre-indexed. |
| 0b100 | For a trapped STC instruction or a trapped T32 LDC instruction this encoding is reserved. |
| 0b110 | For a trapped STC instruction, this encoding is reserved. |

The values 0b101 and 0b111 are reserved. The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE, as described in 'Reserved values in System and memory-mapped registers and translation table entries'.

Bit [2] in this subfield indicates the instruction form, immediate or literal.

Bits [1:0] in this subfield correspond to the bits {P, W} in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|------------------------------------|
| 0b0 | Write to memory. STC instruction. |
| 0b1 | Read from memory. LDC instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following fields describe the configuration settings for the traps that are reported using EC value 0b000110:

- [MDSCR_EL1.TDCC](#), for accesses using AArch32 state, LDC access to [DBGDTRTXint](#) or STC access to [DBGDTRRXint](#) trapped to EL1 or EL2.
- [MDCR_EL2.TDA](#), for accesses using AArch32 state, LDC access to [DBGDTRTXint](#) or STC access to [DBGDTRRXint](#) MCR or MRC access trapped to EL2.
- [MDCR_EL3.TDA](#), for accesses using AArch32 state, LDC access to [DBGDTRTXint](#) or STC access to [DBGDTRRXint](#) MCR or MRC access trapped to EL3.
- If FEAT_FGT is implemented, [MDCR_EL2.TDCC](#) for LDC and STC accesses to the DCC registers at EL0 and EL1 trapped to EL2, and [MDCR_EL3.TDCC](#) for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.

ISS encoding for an exception from an access to SVE, Advanced SIMD or floating-point functionality, resulting from the FPen and TFP traps

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | RES0 | | | | | | | | | | | | | | | | | | | |

The accesses covered by this trap include:

- Execution of SVE or Advanced SIMD and floating-point instructions.
- Accesses to the Advanced SIMD and floating-point System registers.
- Execution of SME instructions.

For an implementation that does not include either SVE or support for Advanced SIMD and floating-point, the exception is reported using the EC value 0b000000.

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

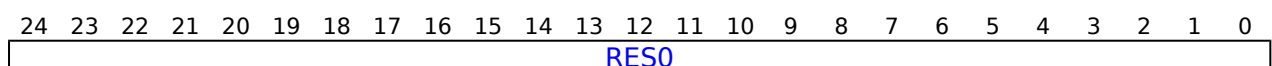
Bits [19:0]

Reserved, RES0.

The following fields describe the configuration settings for the traps that are reported using EC value 0b000111:

- [CPACR_EL1.FPEN](#), for accesses to SIMD and floating-point registers trapped to EL1.
- [CPTR_EL2.FPEN](#) and [CPTR_EL2.TFP](#), for accesses to SIMD and floating-point registers trapped to EL2.
- [CPTR_EL3.TFP](#), for accesses to SIMD and floating-point registers trapped to EL3.

ISS encoding for an exception from an access to SVE functionality, resulting from CPACR_EL1.ZEN, CPTR_EL2.ZEN, CPTR_EL2.TZ, or CPTR_EL3.EZ



The accesses covered by this trap include:

- Execution of SVE instructions when the PE is not in Streaming SVE mode.
- Accesses to the SVE System registers, ZCR_ELx.

For an implementation that does not include SVE, the exception is reported using the EC value 0b000000.

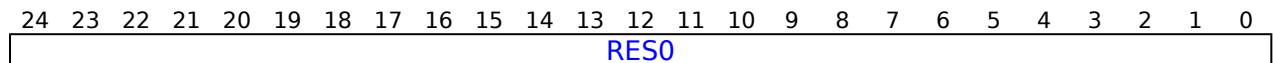
Bits [24:0]

Reserved, RES0.

The following fields describe the configuration settings for the traps that are reported using EC value 0b011001:

- [CPACR_EL1](#).ZEN, for execution of SVE instructions and accesses to SVE registers at EL0 or EL1, trapped to EL1.
- [CPTR_EL2](#).ZEN and [CPTR_EL2](#).TZ, for execution of SVE instructions and accesses to SVE registers at EL0, EL1, or EL2, trapped to EL2.
- [CPTR_EL3](#).EZ, for execution of SVE instructions and accesses to SVE registers from all Exception levels, trapped to EL3.

ISS encoding for an exception from an Illegal Execution state, or a PC or SP alignment fault



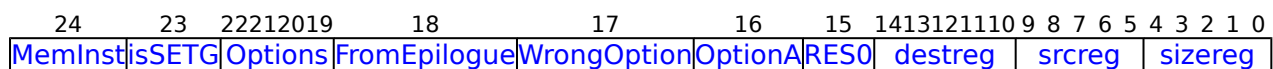
Bits [24:0]

Reserved, RES0.

There are no configuration settings for generating Illegal Execution state exceptions and PC alignment fault exceptions. For more information about these exceptions, see 'The Illegal Execution state exception' and 'PC alignment checking'.

'SP alignment checking' describes the configuration settings for generating SP alignment fault exceptions.

ISS encoding for an exception from the Memory Copy and Memory Set instructions



MemInst, bit [24]

Indicates the memory instruction class causing the exception.

| MemInst | Meaning |
|---------|--|
| 0b0 | CPYFE*, CPYFM*, CPYE*, and CPYM* instructions. |
| 0b1 | SETE*, SETM*, SETGE*, and SETGM* instructions. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

isSETG, bit [23]

Indicates whether the instruction belongs to SETGM* or SETGE* class of instruction.

| isSETG | Meaning |
|---------------|-------------------------------------|
| 0b0 | Not a SETGM* or SETGE* instruction. |
| 0b1 | SETGM* or SETGE* instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Options, bits [22:19]

Options : the Options field of the instruction.

For Memory Copy instructions, bits[22:19] forms the Options field, which holds the bits[15:12] of the instruction.

For Memory Set instructions:

- Bits[22:21] are RES0.
- Bits[20:19] form the Options field, which holds the bits[13:12] of the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FromEpilogue, bit [18]

Indicates whether the instruction belongs to the epilogue class of Memory Copy or Memory Set instructions.

| FromEpilogue | Meaning |
|---------------------|--|
| 0b0 | Not an epilogue instruction. |
| 0b1 | CPYE*, CPYFE*, SETE*, or SETGE* instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WrongOption, bit [17]

Algorithm option.

| WrongOption | Meaning |
|--------------------|-----------------------|
| 0b0 | WrongOption is false. |
| 0b1 | WrongOption is true. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OptionA, bit [16]

Algorithm type indicated by the PSTATE.C bit.

| OptionA | Meaning |
|----------------|-------------------------------------|
| 0b0 | OptionB indicated by PSTATE.C is 0. |
| 0b1 | OptionA indicated by PSTATE.C is 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

destreg, bits [14:10]

The destination register value from the issued instruction, containing the destination address.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

srcreg, bits [9:5]

The source register value from the issued instruction, containing either the source address or the source data.

The reset behavior of this field is:

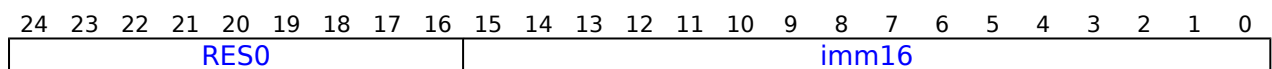
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

sizereg, bits [4:0]

The size register value from the issued instruction, containing the number of bytes to be transferred or set.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from HVC or SVC instruction execution**Bits [24:16]**

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the HVC or SVC instruction.

For an HVC instruction, and for an A64 SVC instruction, this is the value of the imm16 field of the issued instruction.

For an A32 or T32 SVC instruction:

- If the instruction is unconditional, then:
 - For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
 - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- If the instruction is conditional, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

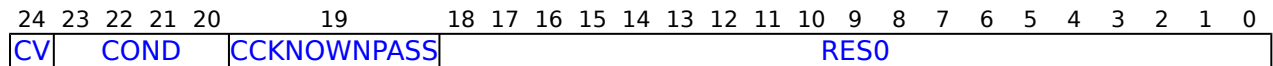
In AArch32 state, the HVC instruction is unconditional, and a conditional SVC instruction generates an exception only if it passes its condition code check. Therefore, the syndrome information for these exceptions does not require conditionality information.

For T32 and A32 instructions, see 'SVC' and 'HVC'.

For A64 instructions, see 'SVC' and 'HVC'.

If FEAT_FGT is implemented, [HFGITR_EL2](#).{SVC_EL1, SVC_EL0} control fine-grained traps on SVC execution.

ISS encoding for an exception from SMC instruction execution in AArch32 state



For an SMC instruction that completes normally and generates an exception that is taken to EL3, the ISS encoding is RES0.

For an SMC instruction that is trapped to EL2 from EL1 because [HCR_EL2.TSC](#) is 1, the ISS encoding is as shown in the diagram.

CV, bit [24]

Condition code valid.

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

For exceptions taken from AArch64, CV is set to 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. See the description of the COND field for more information.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

For exceptions taken from AArch64, this field is set to 0b1110.

The condition code for the trapped instruction. This field is valid only for exceptions taken from AArch32, and only when the value of CV is 1.

For exceptions taken from AArch32:

- When an A32 instruction is trapped, CV is set to 1 and:
 - If the instruction is conditional, COND is set to the condition code field value from the instruction.
 - If the instruction is unconditional, COND is set to 0b1110.
- A conditional A32 instruction that is known to pass its condition code check can be presented either:
 - With COND set to 0b1110, the value for unconditional.
 - With the COND value held in the instruction.
- When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:
 - CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
 - CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.
- For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CCKNOWNPASS, bit [19]

Indicates whether the instruction might have failed its condition code check.

| CCKNOWNPASS | Meaning |
|--------------------|--|
| 0b0 | The instruction was unconditional, or was conditional and passed its condition code check. |
| 0b1 | The instruction was conditional, and might have failed its condition code check. |

Note

In an implementation in which an SMC instruction that fails its code check is not trapped, this field can always return the value 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [18:0]

Reserved, RES0.

[HCR_EL2.TSC](#) describes the configuration settings for trapping SMC instructions to EL2.

'System calls' describes the case where these exceptions are trapped to EL3.

ISS encoding for an exception from SMC instruction execution in AArch64 state

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | imm16 | | | | | | | | | | | | | | | |

Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the issued SMC instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The value of ISS[24:0] described here is used both:

- When an SMC instruction is trapped from EL1 modes.
- When an SMC instruction is not trapped, so completes normally and generates an exception that is taken to EL3.

[HCR_EL2.TSC](#) describes the configuration settings for trapping SMC from EL1 modes.

'System calls' describes the case where these exceptions are trapped to EL3.

ISS encoding for an exception from MSR, MRS, or System instruction execution in AArch64 state

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-----|----|-----|----|-----|----|-----|----|----|----|----|---|---|---|-----|---|---|-----------|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | Op0 | | Op2 | | Op1 | | CRn | | | | Rt | | | | CRm | | | Direction | | | |

Bits [24:22]

Reserved, RES0.

Op0, bits [21:20]

The Op0 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op2, bits [19:17]

The Op2 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Op1, bits [16:14]

The Op1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Rt, bits [9:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|---|
| 0b0 | Write access, including MSR instructions. |
| 0b1 | Read access, including MRS instructions. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For exceptions caused by System instructions, see 'System instructions' subsection of 'Branches, exception generating and System instructions' for the encoding values returned by an instruction.

The following fields describe configuration settings for generating the exception that is reported using EC value 0b011000:

- [SCTLR_EL1](#).UCI, for execution of cache maintenance instructions using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [SCTLR_EL1](#).UCT, for accesses to [CTR_EL0](#) using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [SCTLR_EL1](#).DZE, for execution of DC ZVA instructions using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [SCTLR_EL1](#).UMA, for accesses to the PSTATE interrupt masks using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [CPACR_EL1](#).TTA, for accesses to the trace registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [MDSCR_EL1](#).TDCC, for accesses to the Debug Communications Channel (DCC) registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- If FEAT_FGT is implemented, [MDCR_EL2](#).TDCC for accesses to the DCC registers at EL0 and EL1 trapped to EL2, and [MDCR_EL3](#).TDCC for accesses to the DCC registers at EL0, EL1, and EL2 trapped to EL3.
- [CNTKCTL_EL1](#).{ELOPTEN, EL0VTEN, EL0PCTEN, EL0VCTEN} accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [PMUSERENR_EL0](#).{ER, CR, SW, EN}, for accesses to the Performance Monitor registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [AMUSERENR_EL0](#).EN, for accesses to Activity Monitors registers using AArch64 state, MSR or MRS access trapped to EL1 or EL2.
- [HCR_EL2](#).{TRVM, TVM}, for accesses to virtual memory control registers using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).TDZ, for execution of DC ZVA instructions using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).TTLB, for execution of TLB maintenance instructions using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).{TSW, TPC, TPU}, for execution of cache maintenance instructions using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).TACR, for accesses to the Auxiliary Control Register, [ACTLR_EL1](#), using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).TIDCP, for accesses to lockdown, DMA, and TCM operations using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).{TID1, TID2, TID3}, for accesses to ID group 1, ID group 2 or ID group 3 registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [CPTR_EL2](#).TCPAC, for accesses to [CPACR_EL1](#), using AArch64 state, MSR or MRS access trapped to EL2.
- [CPTR_EL2](#).TTA, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).TTRF, for accesses to the trace filter control register, [TRFCR_EL1](#), using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).TDRA, for accesses to Debug ROM registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).TDOSA, for accesses to powerdown debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- [CNTHCTL_EL2](#).{EL1PCEN, EL1PCTEN}, for accesses to the Generic Timer registers using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).TDA, for accesses to debug registers using AArch64 state, MSR or MRS access trapped to EL2.
- [MDCR_EL2](#).{TPM, TPMCR}, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [CPTR_EL2](#).TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access trapped to EL2.
- [HCR_EL2](#).APK, for accesses to Pointer authentication key registers. using AArch64 state, MSR or MRS access trapped to EL2.

- [HCR_EL2](#).{NV, NV1}, for Nested virtualization register access, using AArch64 state, MSR or MRS access, trapped to EL2.
- [HCR_EL2](#).AT, for execution of AT S1E* instructions, using AArch64 state, MSR or MRS access, trapped to EL2.
- [HCR_EL2](#).{TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access, trapped to EL2.
- [SCR_EL3](#).APK, for accesses to Pointer authentication key registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [SCR_EL3](#).ST, for accesses to the Counter-timer Physical Secure timer registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [SCR_EL3](#).{TERR, FIEN}, for accesses to RAS registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [CPTR_EL3](#).TCPAC, for accesses to [CPTR_EL2](#) and [CPACR_EL1](#) using AArch64 state, MSR or MRS access trapped to EL3.
- [CPTR_EL3](#).TTA, for accesses to the trace registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [MDCR_EL3](#).TTRF, for accesses to the trace filter control registers, [TRFCR_EL1](#) and [TRFCR_EL2](#), using AArch64 state, MSR or MRS access trapped to EL3.
- [MDCR_EL3](#).TDA, for accesses to debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [MDCR_EL3](#).TDOSA, for accesses to powerdown debug registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [MDCR_EL3](#).TPM, for accesses to Performance Monitor registers, using AArch64 state, MSR or MRS access trapped to EL3.
- [CPTR_EL3](#).TAM, for accesses to Activity Monitors registers, using AArch64 state, MSR or MRS access, trapped to EL3.
- If FEAT_EVT is implemented, the following registers control traps for EL1 and EL0 Cache controls that use this EC value:
 - [HCR_EL2](#).{TTLBOS, TTLBIS, TICAB, TOCU, TID4}.
 - [HCR2](#).{TTLBIS, TICAB, TOCU, TID4}.
- If FEAT_FGT is implemented:
 - [SCR_EL3](#).FGTEn, for accesses to the fine-grained trap registers, MSR or MRS access at EL2 trapped to EL3.
 - [HFGTR_EL2](#) for reads and [HFGWTR_EL2](#) for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 trapped to EL2.
 - [HFGITR_EL2](#) for execution of system instructions, MSR or MRS access trapped to EL2
 - [HDFGTR_EL2](#) for reads and [HDFGWTR_EL2](#) for writes of registers, using AArch64 state, MSR or MRS access at EL0 and EL1 state trapped to EL2.
 - [HAFGTR_EL2](#) for reads of Activity Monitor counters, using AArch64 state, MRS access at EL0 and EL1 trapped to EL2.
- If FEAT_RNG_TRAP is implemented:
 - [SCR_EL3](#).TRNDR for reads of [RNDR](#) and [RNDRRS](#) using AArch64 state, MRS access trapped to EL3.
- If FEAT_SME is implemented:
 - [CPTR_EL3](#).ESM, for MSR or MRS accesses to [SMPRI_EL1](#) at EL1, EL2, and EL3, trapped to EL3.
 - [CPTR_EL3](#).ESM, for MSR or MRS accesses to [SMPRIMAP_EL2](#) at EL2 and EL3, trapped to EL3.
 - [SCTLR_EL1](#).EnTP2, for MSR or MRS accesses to [TPIDR2_EL0](#) at EL0, trapped to EL1 or EL2.
 - [SCTLR_EL2](#).EnTP2, for MSR or MRS accesses to [TPIDR2_EL0](#) at EL0, trapped to EL2.
 - [SCR_EL3](#).EnTP2, for MSR or MRS accesses to [TPIDR2_EL0](#) at EL0, EL1, and EL2, trapped to EL3.
- If FEAT_NMI is implemented, [HCRX_EL2](#).TALLINT, for MSR writes of [ALLINT](#) at EL1, trapped to EL2.

ISS encoding for an IMPLEMENTATION DEFINED exception to EL3

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [24:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from an Instruction Abort

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|-----|-----|----|------|-------|------|------|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | SET | FnV | EA | RES0 | S1PTW | RES0 | IFSC | | | | | | |

Bits [24:13]

Reserved, RES0.

SET, bits [12:11]

When FEAT_RAS is implemented:

Synchronous Error Type. When IFSC is 0b010000, describes the PE error state after taking the Instruction Abort exception.

| SET | Meaning |
|------|--------------------------|
| 0b00 | Recoverable state (UER). |
| 0b10 | Uncontainable (UC). |
| 0b11 | Restartable state (UEO). |

All other values are reserved.

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in a PE state that is not recoverable.

This field is valid only if the IFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|---|
| 0b0 | FAR is valid. |
| 0b1 | FAR is not valid, and holds an UNKNOWN value. |

This field is valid only if the IFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

| S1PTW | Meaning |
|--------------|---|
| 0b0 | Fault not on a stage 2 translation for a stage 1 translation table walk. |
| 0b1 | Fault on the stage 2 translation of an access for a stage 1 translation table walk. |

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code.

| IFSC | Meaning | Applies when |
|----------|---|--|
| 0b000000 | Address size fault, level 0 of translation or translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000100 | Translation fault, level 0. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001000 | Access flag fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001100 | Permission fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010000 | Synchronous External abort, not on translation table walk or hardware update of translation table. | |
| 0b010011 | Synchronous External abort on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented |
| 0b010100 | Synchronous External abort on translation table walk or hardware update of translation table, level 0. | |
| 0b010101 | Synchronous External abort on translation table walk or hardware update of translation table, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk or hardware update of translation table, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk or hardware update of translation table, level 3. | |
| 0b011000 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b011011 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented and FEAT_RAS is not implemented |
| 0b011100 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0. | When FEAT_RAS is not implemented |
| 0b011101 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1. | When FEAT_RAS is not implemented |
| 0b011110 | Synchronous parity or ECC error on memory access on translation table walk or | When FEAT_RAS is |

| | | |
|----------|---|---|
| 0b011111 | hardware update of translation table, level 2. Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3. | not implemented When FEAT_RAS is not implemented |
| 0b100011 | Granule Protection Fault on translation table walk or hardware update of translation table, level -1. | When FEAT_RME is implemented and FEAT_LPA2 is implemented |
| 0b100100 | Granule Protection Fault on translation table walk or hardware update of translation table, level 0. | When FEAT_RME is implemented |
| 0b100101 | Granule Protection Fault on translation table walk or hardware update of translation table, level 1. | When FEAT_RME is implemented |
| 0b100110 | Granule Protection Fault on translation table walk or hardware update of translation table, level 2. | When FEAT_RME is implemented |
| 0b100111 | Granule Protection Fault on translation table walk or hardware update of translation table, level 3. | When FEAT_RME is implemented |
| 0b101000 | Granule Protection Fault, not on translation table walk or hardware update of translation table. | When FEAT_RME is implemented |
| 0b101001 | Address size fault, level -1. | When FEAT_LPA2 is implemented |
| 0b101011 | Translation fault, level -1. | When FEAT_LPA2 is implemented |
| 0b110000 | TLB conflict abort. | |
| 0b110001 | Unsupported atomic hardware update fault. | When FEAT_HAFDBS is implemented |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults'.

Note

Because Access flag faults and Permission faults can result only from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception due to SME functionality

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|------|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | SMTc | | |

The accesses covered by this trap include:

- Execution of SME instructions.
- Execution of SVE and Advanced SIMD instructions, when the PE is in Streaming SVE mode.
- Direct accesses of [SVCR](#), [SMCR_EL1](#), [SMCR_EL2](#), [SMCR_EL3](#).

Bits [24:3]

Reserved, RES0.

SMTC, bits [2:0]

SME Trap Code. Identifies the reason for instruction trapping.

| SMTC | Meaning |
|-------|--|
| 0b000 | Access to SME functionality trapped as a result of CPACR_EL1 .SMEN, CPTR_EL2 .SMEN, CPTR_EL2 .TSM, or CPTR_EL3 .ESM, that is not reported using EC 0b000000. |
| 0b001 | Advanced SIMD, SVE, or SVE2 instruction trapped because PSTATE.SM is 1. |
| 0b010 | SME instruction trapped because PSTATE.SM is 0. |
| 0b011 | SME instruction trapped because PSTATE.ZA is 0. |

All other values are reserved.

The following fields describe the configuration settings for the traps that are reported using the EC value 0b011101:

- [CPACR_EL1](#).SMEN, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access [SVCR](#) and [SMCR_EL1](#) System registers at EL1 and EL0, trapped to EL1 or EL2.
- [CPTR_EL2](#).SMEN and [CPTR_EL2](#).TSM, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access [SVCR](#), [SMCR_EL1](#), [SMCR_EL2](#) at EL2, EL1, or EL0, trapped to EL2.
- [CPTR_EL3](#).ESM, for execution of SME instructions, SVE instructions when the PE is in Streaming SVE mode, and instructions that directly access [SVCR](#), [SMCR_EL1](#), [SMCR_EL2](#), [SMCR_EL3](#) from all Exception levels and any Security state, trapped to EL3.

ISS encoding for an exception from a Granule Protection Check

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|-------|----|-----|-------|----|----|------|----|------|------|----|-------|-----|------|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | S2PTW | | InD | GPCSC | | | VNCR | | RES0 | RES0 | CM | S1PTW | WnR | xFSC | | | | | | | | |

Bits [24:22]

Reserved, RES0.

S2PTW, bit [21]

Indicates whether the Granule Protection Check exception was on an access made for a stage 2 translation table walk.

| S2PTW | Meaning |
|-------|--|
| 0b0 | Fault not on a stage 2 translation table walk. |
| 0b1 | Fault on a stage 2 translation table walk. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

InD, bit [20]

Indicates whether the Granule Protection Check exception was on an instruction or data access.

| InD | Meaning |
|-----|---------------------|
| 0b0 | Data access. |
| 0b1 | Instruction access. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GPCSC, bits [19:14]

Granule Protection Check Status Code.

| GPCSC | Meaning |
|----------|---|
| 0b000000 | GPT address size fault at level 0. |
| 0b000001 | GPT address size fault at level 1. |
| 0b000100 | GPT walk fault at level 0. |
| 0b000101 | GPT walk fault at level 1. |
| 0b001100 | Granule protection fault at level 0. |
| 0b001101 | Granule protection fault at level 1. |
| 0b010100 | Synchronous External abort on GPT fetch at level 0. |
| 0b010101 | Synchronous External abort on GPT fetch at level 1. |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VNCR, bit [13]

When FEAT_NV2 is implemented:

Indicates that the fault came from use of [VNCR_EL2](#) register by EL1 code.

| VNCR | Meaning |
|------|--|
| 0b0 | The fault was not generated by the use of VNCR_EL2 , by an MRS or MSR instruction executed at EL1. |
| 0b1 | The fault was generated by the use of VNCR_EL2 , by an MRS or MSR instruction executed at EL1. |

This field is 0 in ESR_EL1.

When InD is '1', this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [12:11]

Reserved, RES0.

Bits [10:9]

Reserved, RES0.

CM, bit [8]

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:

| CM | Meaning |
|-----------|--|
| 0b0 | The Data Abort was not generated by the execution of one of the System instructions identified in the description of value 1. |
| 0b1 | The Data Abort was generated by either the execution of a cache maintenance instruction or by a synchronous fault on the execution of an address translation instruction. The DC ZVA , DC GVA , and DC GZVA instructions are not classified as cache maintenance instructions, and therefore their execution cannot cause this field to be set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

Indicates whether the Granule Protection Check exception was on an access for stage 2 translation for a stage 1 translation table walk:

| S1PTW | Meaning |
|--------------|---|
| 0b0 | Fault not on a stage 2 translation for a stage 1 translation table walk. |
| 0b1 | Fault on the stage 2 translation of an access for a stage 1 translation table walk. |

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

| WnR | Meaning |
|------------|--|
| 0b0 | Abort caused by an instruction reading from a memory location. |
| 0b1 | Abort caused by an instruction writing to a memory location. |

When InD is '1', this field is RES0.

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is UNKNOWN for:

- An External abort on an Atomic access.
- A fault reported using a DFSC value of 0b110101 or 0b110001, indicating an unsupported Exclusive or atomic access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

xFSC, bits [5:0]

Instruction or Data Fault Status Code.

| xFSC | Meaning | Applies when |
|-------------|---|---|
| 0b100011 | Granule Protection Fault on translation table walk or hardware update of translation table, level -1. | When FEAT_RME is implemented and FEAT_LPA2 is implemented |
| 0b100100 | Granule Protection Fault on translation table walk or hardware update of translation table, level 0. | When FEAT_RME is implemented |
| 0b100101 | Granule Protection Fault on translation table walk or hardware update of translation table, level 1. | When FEAT_RME is implemented |
| 0b100110 | Granule Protection Fault on translation table walk or hardware update of translation table, level 2. | When FEAT_RME is implemented |
| 0b100111 | Granule Protection Fault on translation table walk or hardware update of translation table, level 3. | When FEAT_RME is implemented |
| 0b101000 | Granule Protection Fault, not on translation table walk or hardware update of translation table. | When FEAT_RME is implemented |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults'.

Note

Because Access flag faults and Permission faults can result only from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from a Data Abort

| | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|-----|-----|----|----|----|-----|----|----|----|----|------|-------------|-----|----|----|-------|-----|---|---|---|---|---|------|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ISV | SAS | SSE | | | | SRT | | | SF | AR | VNCR | Bits[12:11] | FnV | EA | CM | S1PTW | WnR | | | | | | DFSC | |

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV or ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this ISS encoding includes ISS2, bits[36:32].

ISV, bit [24]

Instruction Syndrome Valid. Indicates whether the syndrome information in ISS[23:14] is valid.

| ISV | Meaning |
|-----|---|
| 0b0 | No valid instruction syndrome. ISS[23:14] are RES0. |
| 0b1 | ISS[23:14] hold a valid instruction syndrome. |

In ESR_EL2, ISV is 1 when FEAT_LS64 is implemented and a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

For other faults reported in ESR_EL2, ISV is 0 except for the following stage 2 aborts:

- AArch64 loads and stores of a single general-purpose register (including the register specified with 0b11111, including those with Acquire/Release semantics, but excluding Load Exclusive or Store Exclusive and excluding those with writeback).
- AArch32 instructions where the instruction:
 - Is an LDR, LDA, LDRT, LDRSH, LDRSHT, LDRH, LDAH, LDRHT, LDRSB, LDRSBT, LDRB, LDAB, LDRBT, STR, STL, STRT, STRH, STLH, STRHT, STRB, STLB, or STRBT instruction.
 - Is not performing register writeback.
 - Is not using R15 as a source or destination register.

For these stage 2 aborts, ISV is UNKNOWN if the exception was generated in Debug state in memory access mode, and otherwise indicates whether ISS[23:14] hold a valid syndrome.

For faults reported in ESR_EL1 or ESR_EL3, ISV is 1 when FEAT_LS64 is implemented and a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault. ISV is 0 for all other faults reported in ESR_EL1 or ESR_EL3.

When FEAT_RAS is implemented, ISV is 0 for any synchronous External abort.

For ISS reporting, a stage 2 abort on a stage 1 translation table walk does not return a valid instruction syndrome, and therefore ISV is 0 for these aborts.

When FEAT_RAS is not implemented, it is IMPLEMENTATION DEFINED whether ISV is set to 1 or 0 on a synchronous External abort on a stage 2 translation table walk.

When FEAT_MTE2 is implemented, for a synchronous Tag Check Fault abort taken to ELx, ESR_ELx.FNV is 0 and FAR_ELx is valid.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SAS, bits [23:22]

When ISV == 1:

Syndrome Access Size. Indicates the size of the access attempted by the faulting operation.

| SAS | Meaning |
|------|------------|
| 0b00 | Byte |
| 0b01 | Halfword |
| 0b10 | Word |
| 0b11 | Doubleword |

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0b11.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSE, bit [21]**When ISV == 1:**

Syndrome Sign Extend. For a byte, halfword, or word load operation, indicates whether the data item must be sign extended.

| SSE | Meaning |
|-----|----------------------------------|
| 0b0 | Sign-extension not required. |
| 0b1 | Data item must be sign-extended. |

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

For all other operations, this field is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SRT, bits [20:16]**When ISV == 1:**

Syndrome Register Transfer. The register number of the Wt/Xt/Rt operand of the faulting instruction.

If the exception was taken from an Exception level that is using AArch32, then this is the AArch64 view of the register. See 'Mapping of the general-purpose registers between the Execution states'.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SF, bit [15]**When ISV == 1:**

Width of the register accessed by the instruction is Sixty-Four.

| SF | Meaning |
|-----|--|
| 0b0 | Instruction loads/stores a 32-bit wide register. |
| 0b1 | Instruction loads/stores a 64-bit wide register. |

Note

This field specifies the register width identified by the instruction, not the Execution state.

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 1.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AR, bit [14]

When ISV == 1:

Acquire/Release.

| AR | Meaning |
|-----|---|
| 0b0 | Instruction did not have acquire/release semantics. |
| 0b1 | Instruction did have acquire/release semantics. |

When FEAT_LS64 is implemented, if a memory access generated by an ST64BV, ST64BV0, ST64B, or LD64B instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault, then this field is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VNCR, bit [13]

When FEAT_NV2 is implemented:

Indicates that the fault came from use of [VNCR_EL2](#) register by EL1 code.

| VNCR | Meaning |
|------|--|
| 0b0 | The fault was not generated by the use of VNCR_EL2 , by an MRS or MSR instruction executed at EL1. |
| 0b1 | The fault was generated by the use of VNCR_EL2 , by an MRS or MSR instruction executed at EL1. |

This field is 0 in ESR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SET, bits [12:11]

When FEAT_RAS is implemented and FEAT_LS64 is not implemented:

Synchronous Error Type. When DFSC is 0b010000, describes the PE error state after taking the Data Abort exception.

| SET | Meaning |
|------|--------------------------|
| 0b00 | Recoverable state (UER). |
| 0b10 | Uncontainable (UC). |
| 0b11 | Restartable state (UEO). |

All other values are reserved.

Note

Software can use this information to determine what recovery might be possible. Taking a synchronous External Abort exception might result in a PE state that is not recoverable.

This field is valid only if the DFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_LS64 is implemented:

Load/Store Type. Used when an LD64B, ST64B, ST64BV, or ST64BV0 instruction generates a Data Abort for a Translation fault, Access flag fault, or Permission fault.

| LST | Meaning |
|------|---|
| 0b01 | An ST64BV instruction generated the Data Abort. |
| 0b10 | An LD64B or ST64B instruction generated the Data Abort. |
| 0b11 | An ST64BV0 instruction generated the Data Abort. |

All other values are reserved.

This field is valid only if the DFSC code is 0b110101. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|---|
| 0b0 | FAR is valid. |
| 0b1 | FAR is not valid, and holds an UNKNOWN value. |

This field is valid only if the DFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CM, bit [8]

Cache maintenance. Indicates whether the Data Abort came from a cache maintenance or address translation instruction:

| CM | Meaning |
|-----|--|
| 0b0 | The Data Abort was not generated by the execution of one of the System instructions identified in the description of value 1. |
| 0b1 | The Data Abort was generated by either the execution of a cache maintenance instruction or by a synchronous fault on the execution of an address translation instruction. The DC ZVA , DC GVA , and DC GZVA instructions are not classified as cache maintenance instructions, and therefore their execution cannot cause this field to be set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

| S1PTW | Meaning |
|-------|---|
| 0b0 | Fault not on a stage 2 translation for a stage 1 translation table walk. |
| 0b1 | Fault on the stage 2 translation of an access for a stage 1 translation table walk. |

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

| WnR | Meaning |
|-----|--|
| 0b0 | Abort caused by an instruction reading from a memory location. |
| 0b1 | Abort caused by an instruction writing to a memory location. |

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

For faults from an atomic instruction that both reads and writes from a memory location, this bit is set to 0 if a read of the address specified by the instruction would have generated the fault which is being reported, otherwise it is set to 1. The architecture permits, but does not require, a relaxation of this requirement such that for all stage 2 aborts on stage 1 translation table walks for atomic instructions, the WnR bit is always 0.

This field is UNKNOWN for:

- An External abort on an Atomic access.
- A fault reported using a DFSC value of 0b110101 or 0b110001, indicating an unsupported Exclusive or atomic access.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code.

| DFSC | Meaning | Applies when |
|----------|---|--|
| 0b000000 | Address size fault, level 0 of translation or translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000100 | Translation fault, level 0. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001000 | Access flag fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001100 | Permission fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010000 | Synchronous External abort, not on translation table walk or hardware update of translation table. | |
| 0b010001 | Synchronous Tag Check Fault. | When FEAT_MTE2 is implemented |
| 0b010011 | Synchronous External abort on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented |
| 0b010100 | Synchronous External abort on translation table walk or hardware update of translation table, level 0. | |
| 0b010101 | Synchronous External abort on translation table walk or hardware update of translation table, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk or hardware update of translation table, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk or hardware update of translation table, level 3. | |
| 0b011000 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b011011 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented and FEAT_RAS is not implemented |
| 0b011100 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0. | When FEAT_RAS is not implemented |
| 0b011101 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1. | When FEAT_RAS is not implemented |

| | | |
|-----------|--|---|
| 0b0111110 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2. | When FEAT_RAS is not implemented |
| 0b0111111 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3. | When FEAT_RAS is not implemented |
| 0b100001 | Alignment fault. | |
| 0b100011 | Granule Protection Fault on translation table walk or hardware update of translation table, level -1. | When FEAT_RME is implemented and FEAT_LPA2 is implemented |
| 0b100100 | Granule Protection Fault on translation table walk or hardware update of translation table, level 0. | When FEAT_RME is implemented |
| 0b100101 | Granule Protection Fault on translation table walk or hardware update of translation table, level 1. | When FEAT_RME is implemented |
| 0b100110 | Granule Protection Fault on translation table walk or hardware update of translation table, level 2. | When FEAT_RME is implemented |
| 0b100111 | Granule Protection Fault on translation table walk or hardware update of translation table, level 3. | When FEAT_RME is implemented |
| 0b101000 | Granule Protection Fault, not on translation table walk or hardware update of translation table. | When FEAT_RME is implemented |
| 0b101001 | Address size fault, level -1. | When FEAT_LPA2 is implemented |
| 0b101011 | Translation fault, level -1. | When FEAT_LPA2 is implemented |
| 0b110000 | TLB conflict abort. | |
| 0b110001 | Unsupported atomic hardware update fault. | When FEAT_HAFDBS is implemented |
| 0b110100 | IMPLEMENTATION DEFINED fault (Lockdown). | |
| 0b110101 | IMPLEMENTATION DEFINED fault (Unsupported Exclusive or Atomic access). | |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults'.

Note

Because Access flag faults and Permission faults can result only from a Block or Page translation table descriptor, they cannot occur at level 0.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS encoding for an exception from a trapped floating-point exception

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|------|----|----|----|----|----|----|----|----|----|--------|----|----|-----|------|-----|-----|-----|-----|-----|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | TFV | RES0 | | | | | | | | | | VECITR | | | IDF | RES0 | IXF | UFF | OFF | DZF | IOF | | | |

Bit [24]

Reserved, RES0.

TFV, bit [23]

Trapped Fault Valid bit. Indicates whether the IDF, IXF, UFF, OFF, DZF, and IOF bits hold valid information about trapped floating-point exceptions.

| TFV | Meaning |
|-----|---|
| 0b0 | The IDF, IXF, UFF, OFF, DZF, and IOF bits do not hold valid information about trapped floating-point exceptions and are UNKNOWN. |
| 0b1 | One or more floating-point exceptions occurred during an operation performed while executing the reported instruction. The IDF, IXF, UFF, OFF, DZF, and IOF bits indicate trapped floating-point exceptions that occurred. For more information, see 'Floating-point exceptions and exception traps'. |

It is IMPLEMENTATION DEFINED whether this field is set to 0 on an exception generated by a trapped floating-point exception from an instruction that is performing floating-point operations on more than one lane of a vector.

Note

This is not a requirement. Implementations can set this field to 1 on a trapped floating-point exception from an instruction and return valid information in the {IDF, IXF, UFF, OFF, DZF, IOF} fields.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [22:11]

Reserved, RES0.

VECITR, bits [10:8]

For a trapped floating-point exception from an instruction executed in AArch32 state this field is RES1.

For a trapped floating-point exception from an instruction executed in AArch64 state this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| IDF | Meaning |
|-----|--|
| 0b0 | Input denormal floating-point exception has not occurred. |
| 0b1 | Input denormal floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| IXF | Meaning |
|-----|---|
| 0b0 | Inexact floating-point exception has not occurred. |
| 0b1 | Inexact floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFF, bit [3]

Underflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| UFF | Meaning |
|-----|---|
| 0b0 | Underflow floating-point exception has not occurred. |
| 0b1 | Underflow floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFF, bit [2]

Overflow floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| OFF | Meaning |
|-----|--|
| 0b0 | Overflow floating-point exception has not occurred. |
| 0b1 | Overflow floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZF, bit [1]

Divide by Zero floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| DZF | Meaning |
|-----|--|
| 0b0 | Divide by Zero floating-point exception has not occurred. |
| 0b1 | Divide by Zero floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOF, bit [0]

Invalid Operation floating-point exception trapped bit. If the TFV field is 0, this bit is UNKNOWN. Otherwise, the possible values of this bit are:

| IOF | Meaning |
|-----|---|
| 0b0 | Invalid Operation floating-point exception has not occurred. |
| 0b1 | Invalid Operation floating-point exception occurred during execution of the reported instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

In an implementation that supports the trapping of floating-point exceptions:

- From an Exception level using AArch64, the [FPCR](#).{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.
- From an Exception level using AArch32, the [FPSCR](#).{IDE, IXE, UFE, OFE, DZE, IOE} bits enable each of the floating-point exception traps.

ISS encoding for an SError interrupt

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|------|----|----|----|----|----|----|----|----|----|------|----|-----|---|----|---|------|---|---|------|---|---|---|--|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| IDS | | RES0 | | | | | | | | | | IESB | | AET | | EA | | RES0 | | | DFSC | | | | |

IDS, bit [24]

IMPLEMENTATION DEFINED syndrome.

| IDS | Meaning |
|---|---|
| 0b0 | Bits [23:0] of the ISS field holds the fields described in this encoding. |
| Note If FEAT_RAS is not implemented, bits [23:0] of the ISS field are RES0. | |
| 0b1 | Bits [23:0] of the ISS field holds IMPLEMENTATION DEFINED syndrome information that can be used to provide additional information about the SError interrupt. |

Note

This field was previously called ISV.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:14]

Reserved, RES0.

IESB, bit [13]

When **FEAT_IESB** is implemented:

Implicit error synchronization event.

| IESB | Meaning |
|-------------|--|
| 0b0 | The SError interrupt was either not synchronized by the implicit error synchronization event or not taken immediately. |
| 0b1 | The SError interrupt was synchronized by the implicit error synchronization event and taken immediately. |

This field is valid only if the DFSC code is 0b010001. It is RES0 for all other errors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AET, bits [12:10]

When **FEAT_RAS** is implemented:

Asynchronous Error Type.

When DFSC is 0b010001, describes the PE error state after taking the SError interrupt exception.

| AET | Meaning |
|------------|----------------------------|
| 0b000 | Uncontainable (UC). |
| 0b001 | Unrecoverable state (UEU). |
| 0b010 | Restartable state (UEO). |
| 0b011 | Recoverable state (UER). |
| 0b110 | Corrected (CE). |

All other values are reserved.

If multiple errors are taken as a single SError interrupt exception, the overall PE error state is reported.

Note

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

This field is valid only if the DFSC code is 0b010001. It is RES0 for all other errors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EA, bit [9]**When FEAT_RAS is implemented:**

External abort type. When DFSC is 0b010001, provides an IMPLEMENTATION DEFINED classification of External aborts.

This field is valid only if the DFSC code is 0b010001. It is RES0 for all other errors.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [8:6]

Reserved, RES0.

DFSC, bits [5:0]**When FEAT_RAS is implemented:**

Data Fault Status Code.

| DFSC | Meaning |
|----------|--------------------------------|
| 0b000000 | Uncategorized error. |
| 0b010001 | Asynchronous SError interrupt. |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ISS encoding for an exception from a Breakpoint or Vector Catch debug exception

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|------|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | IFSC | | | | |

Bits [24:6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code.

| IFSC | Meaning |
|----------|------------------|
| 0b100010 | Debug exception. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions:

- For exceptions from AArch64, see 'Breakpoint exceptions'.
- For exceptions from AArch32, see 'Breakpoint exceptions' and 'Vector Catch exceptions'.

ISS encoding for an exception from a Software Step exception

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|----|---|------|---|---|---|--|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| ISV | | RES0 | | | | | | | | | | | | | | | | | EX | | IFSC | | | | |

ISV, bit [24]

Instruction syndrome valid. Indicates whether the EX bit, ISS[6], is valid, as follows:

| ISV | Meaning |
|-----|------------------|
| 0b0 | EX bit is RES0. |
| 0b1 | EX bit is valid. |

See the EX bit description for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:7]

Reserved, RES0.

EX, bit [6]

Exclusive operation. If the ISV bit is set to 1, this bit indicates whether a Load-Exclusive instruction was stepped.

| EX | Meaning |
|-----|---|
| 0b0 | An instruction other than a Load-Exclusive instruction was stepped. |
| 0b1 | A Load-Exclusive instruction was stepped. |

If the ISV bit is set to 0, this bit is RES0, indicating no syndrome data is available.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IFSC, bits [5:0]

Instruction Fault Status Code.

| IFSC | Meaning |
|----------|------------------|
| 0b100010 | Debug exception. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Software Step exceptions'.

ISS encoding for an exception from a Watchpoint exception

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----------|----|------|----|----|---|-----------|---|------|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | RES0VNCR | | RES0 | | | | CMRES0WnR | | DFSC | | | | | | |

Bits [24:15]

Reserved, RES0.

Bit [14]

Reserved, RES0.

VNCR, bit [13]

When FEAT_NV2 is implemented:

Indicates that the watchpoint came from use of [VNCR_EL2](#) register by EL1 code.

| VNCR | Meaning |
|------|--|
| 0b0 | The watchpoint was not generated by the use of VNCR_EL2 by EL1 code. |
| 0b1 | The watchpoint was generated by the use of VNCR_EL2 by EL1 code. |

This field is 0 in ESR_EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [12:9]

Reserved, RES0.

CM, bit [8]

Cache maintenance. Indicates whether the Watchpoint exception came from a cache maintenance or address translation instruction:

| CM | Meaning |
|-----|---|
| 0b0 | The Watchpoint exception was not generated by the execution of one of the System instructions identified in the description of value 1. |
| 0b1 | The Watchpoint exception was generated by either the execution of a cache maintenance instruction or by a synchronous Watchpoint exception on the execution of an address translation instruction. The DC ZVA , DC GVA , and DC GZVA instructions are not classified as a cache maintenance instructions, and therefore their execution cannot cause this field to be set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

WnR, bit [6]

Write not Read. Indicates whether the Watchpoint exception was caused by an instruction writing to a memory location, or by an instruction reading from a memory location.

| WnR | Meaning |
|-----|---|
| 0b0 | Watchpoint exception caused by an instruction reading from a memory location. |
| 0b1 | Watchpoint exception caused by an instruction writing to a memory location. |

For Watchpoint exceptions on cache maintenance and address translation instructions, this bit always returns a value of 1.

For Watchpoint exceptions from an atomic instruction, this field is set to 0 if a read of the location would have generated the Watchpoint exception, otherwise it is set to 1.

If multiple watchpoints match on the same access, it is UNPREDICTABLE which watchpoint generates the Watchpoint exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code.

| DFSC | Meaning |
|----------|------------------|
| 0b100010 | Debug exception. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Watchpoint exceptions'.

ISS encoding for an exception from execution of a Breakpoint instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | Comment | | | | | | | | | | | | | | |

Bits [24:16]

Reserved, RES0.

Comment, bits [15:0]

Set to the instruction comment field value, zero extended as necessary.

For the AArch32 BKPT instructions, the comment field is described as the immediate field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see 'Breakpoint instruction exceptions'.

ISS encoding for an exception from an ERET, ERETAA, or ERETAB instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|------|---|-------|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | ERET | | ERETA |

This EC value applies when FEAT_FGT is implemented, or when [HCR_EL2.NV](#) is 1.

Bits [24:2]

Reserved, RES0.

ERET, bit [1]

Indicates whether an ERET or ERETA* instruction was trapped to EL2.

| ERET | Meaning |
|------|--|
| 0b0 | ERET instruction trapped to EL2. |
| 0b1 | ERETAA or ERETAB instruction trapped to EL2. |

If this bit is 0, the ERETA field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ERETA, bit [0]

Indicates whether an ERETAA or ERETAB instruction was trapped to EL2.

| ERETA | Meaning |
|-------|------------------------------------|
| 0b0 | ERETAA instruction trapped to EL2. |
| 0b1 | ERETAB instruction trapped to EL2. |

When the ERET field is 0, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

For more information about generating these exceptions, see [HCR_EL2.NV](#).

If FEAT_FGT is implemented, [HFGITR_EL2](#).ERET controls fine-grained trap exceptions from ERET, ERETAA and ERETAB execution.

ISS encoding for an exception from a TSTART instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|------|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | Rd | | | RES0 | | | | | | |

Bits [24:10]

Reserved, RES0.

Rd, bits [9:5]

The Rd value from the issued instruction, the general purpose register used for the destination.

Bits [4:0]

Reserved, RES0.

ISS encoding for an exception from Branch Target Identification instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|--------|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | BTTYPE | |

Bits [24:2]

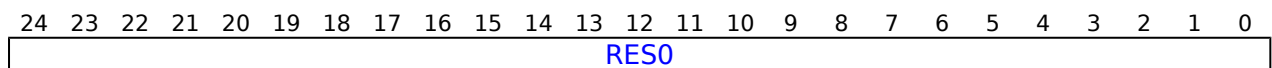
Reserved, RES0.

BTYPE, bits [1:0]

This field is set to the PSTATE.BTYPE value that generated the Branch Target Exception.

For more information about generating these exceptions, see 'The AArch64 application level programmers model'.

ISS encoding for an exception from a Pointer Authentication instruction when HCR_EL2.API == 0 || SCR_EL3.API == 0

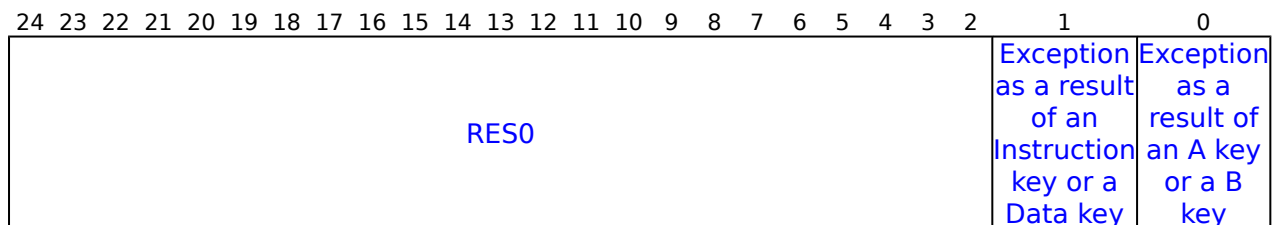
**Bits [24:0]**

Reserved, RES0.

For more information about generating these exceptions, see:

- [HCR_EL2.API](#), for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL2.
- [SCR_EL3.API](#), for exceptions from Pointer authentication instructions, using AArch64 state, trapped to EL3.

ISS encoding for an exception from a Pointer Authentication instruction authentication failure

**Bits [24:2]**

Reserved, RES0.

Bit [1]

This field indicates whether the exception is as a result of an Instruction key or a Data key.

| Meaning | |
|---------|------------------|
| 0b0 | Instruction Key. |
| 0b1 | Data Key. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

This field indicates whether the exception is as a result of an A key or a B key.

| | Meaning |
|-----|---------|
| 0b0 | A key. |
| 0b1 | B key. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following instructions generate an exception when the Pointer Authentication Code (PAC) is incorrect:

- AUTIASP, AUTIAZ, AUTIA1716.
- AUTIBSP, AUTIBZ, AUTIB1716.
- AUTIA, AUTDA, AUTIB, AUTDB.
- AUTIZA, AUTIZB, AUTDZA, AUTDZB.

It is IMPLEMENTATION DEFINED whether the following instructions generate an exception directly from the authorization failure, rather than changing the address in a way that will generate a Translation fault when the address is accessed:

- RETAA, RETAB.
- BRAA, BRAB, BLRAA, BLRAB.
- BRAAZ, BRABZ, BLRAAZ, BLRABZ.
- ERETA, ERETAB.
- LDRAA, LDRAB, whether the authenticated address is written back to the base register or not.

Accessing ESR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ESR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0101 | 0b0010 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return ESR_EL3;
```

MSR ESR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0101 | 0b0010 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    ESR_EL3 = X[t];
```


FAR_EL1, Fault Address Register (EL1)

The FAR_EL1 characteristics are:

Purpose

Holds the faulting Virtual Address for all synchronous Instruction or Data Abort, PC alignment fault and Watchpoint exceptions that are taken to EL1.

Configuration

AArch64 System register FAR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DFAR\[31:0\]](#) (NS).

AArch64 System register FAR_EL1 bits [63:32] are architecturally mapped to AArch32 System register [IFAR\[31:0\]](#) (NS).

Attributes

FAR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Faulting Virtual Address for synchronous exceptions taken to EL1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Faulting Virtual Address for synchronous exceptions taken to EL1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Faulting Virtual Address for synchronous exceptions taken to EL1. Exceptions that set the FAR_EL1 are Instruction Aborts (EC 0x20 or 0x21), Data Aborts (EC 0x24 or 0x25), PC alignment faults (EC 0x22), and Watchpoints (EC 0x34 or 0x35). [ESR_EL1](#).EC holds the EC value for the exception.

For a synchronous External abort, if the VA that generated the abort was from an address range for which $\text{TCR_ELx.TBI}\{<0|1>\} == 1$ for the translation regime in use when the abort was generated, then the top eight bits of FAR_EL1 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if [ESR_EL1](#).FnV is 0, and the FAR_EL1 is UNKNOWN if [ESR_EL1](#).FnV is 1.

For all other exceptions taken to EL1, the FAR_EL1 is UNKNOWN.

If a memory fault that sets FAR_EL1, other than a Tag Check Fault, is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

On an exception due to a Tag Check Fault caused by a data cache maintenance or other DC instruction, the address held in FAR_EL1 is IMPLEMENTATION DEFINED as one of the following:

- The lowest address that gave rise to the fault.
- The address specified in the register argument of the instruction as generated by MMU faults caused by [DC ZVA](#).

If the exception that updates FAR_EL1 is taken from an Exception level that is using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_EL1 are 0x00000001:

- The faulting address was generated by a load or store instruction that sequentially incremented from address 0xFFFFFFFF. Such a load or store is CONSTRAINED UNPREDICTABLE.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

For a Data Abort or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see 'Address tagging in AArch64 state'.

For a synchronous Tag Check Fault abort, bits[63:60] are UNKNOWN.

Execution at EL0 makes FAR_EL1 become UNKNOWN.

Note

The address held in this field is an address accessed by the instruction fetch or data access that caused the exception that gave rise to the instruction or data abort. It is the lower address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores an unaligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

FAR_EL1 is made UNKNOWN on an exception return from EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing FAR_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic FAR_EL1 or FAR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, FAR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.FAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x220];
    else
        return FAR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return FAR_EL2;
    else
        return FAR_EL1;
elsif PSTATE.EL == EL3 then
    return FAR_EL1;

```

MSR FAR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.FAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x220] = X[t];
    else
        FAR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' then
            FAR_EL2 = X[t];
        else
            FAR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        FAR_EL1 = X[t];

```

MRS <Xt>, FAR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x220];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' then
            return FAR_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
            return FAR_EL1;
        else
            UNDEFINED;

```

MSR FAR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x220] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        FAR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        FAR_EL1 = X[t];
    else
        UNDEFINED;

```

MRS <Xt>, FAR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return FAR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return FAR_EL2;
elsif PSTATE.EL == EL3 then
    return FAR_EL2;

```

MSR FAR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        FAR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    FAR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    FAR_EL2 = X[t];

```


FAR_EL2, Fault Address Register (EL2)

The FAR_EL2 characteristics are:

Purpose

Holds the faulting Virtual Address for all synchronous Instruction or Data Abort, PC alignment fault and Watchpoint exceptions that are taken to EL2.

Configuration

AArch64 System register FAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HDFAR\[31:0\]](#).

AArch64 System register FAR_EL2 bits [63:32] are architecturally mapped to AArch32 System register [HIFAR\[31:0\]](#).

AArch64 System register FAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [DFAR\[31:0\]](#) (S) when EL2 is implemented.

AArch64 System register FAR_EL2 bits [63:32] are architecturally mapped to AArch32 System register [IFAR\[31:0\]](#) (S) when EL2 is implemented.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

FAR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Faulting Virtual Address for synchronous exceptions taken to EL2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Faulting Virtual Address for synchronous exceptions taken to EL2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Faulting Virtual Address for synchronous exceptions taken to EL2. Exceptions that set the FAR_EL2 are Instruction Aborts (EC 0x20 or 0x21), Data Aborts (EC 0x24 or 0x25), PC alignment faults (EC 0x22), and Watchpoints (EC 0x34 or 0x35). [ESR_EL2](#).EC holds the EC value for the exception.

For a synchronous External abort, if the VA that generated the abort was from an address range for which $\text{TCR_ELx.TBI}\{\text{<0|1>}\} == 1$ for the translation regime in use when the abort was generated, then the top eight bits of FAR_EL2 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if [ESR_EL2](#).FnV is 0, and the FAR_EL2 is UNKNOWN if [ESR_EL2](#).FnV is 1.

For all other exceptions taken to EL2, the FAR_EL2 is UNKNOWN.

If a memory fault that sets FAR_EL2, other than a Tag Check Fault, is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

On an exception due to a Tag Check Fault caused by a data cache maintenance or other DC instruction, the address held in FAR_EL2 is IMPLEMENTATION DEFINED as one of the following:

- The lowest address that gave rise to the fault.
- The address specified in the register argument of the instruction as generated by MMU faults caused by [DC ZVA](#).

If the exception that updates FAR_EL2 is taken from an Exception level that is using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_ELx are 0x00000001:

- The faulting address was generated by a load or store instruction that sequentially incremented from address 0xFFFFFFFF. Such a load or store instruction is CONSTRAINED UNPREDICTABLE.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

For a Data Abort or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see 'Address tagging in AArch64 state'.

For a synchronous Tag Check Fault abort, bits[63:60] are UNKNOWN.

Execution at EL1 or EL0 makes FAR_EL2 become UNKNOWN.

Note

The address held in this field is an address accessed by the instruction fetch or data access that caused the exception that gave rise to the instruction or data abort. It is the lower address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores an unaligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

FAR_EL2 is made UNKNOWN on an exception return from EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing FAR_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic FAR_EL2 or FAR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, FAR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return FAR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return FAR_EL2;
elsif PSTATE.EL == EL3 then
    return FAR_EL2;

```

MSR FAR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        FAR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    FAR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    FAR_EL2 = X[t];

```

MRS <Xt>, FAR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.FAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x220];
    else
        return FAR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return FAR_EL2;
    else
        return FAR_EL1;
elsif PSTATE.EL == EL3 then
    return FAR_EL1;

```

MSR FAR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.FAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x220] = X[t];
    else
        FAR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        FAR_EL2 = X[t];
    else
        FAR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    FAR_EL1 = X[t];

```


FAR_EL3, Fault Address Register (EL3)

The FAR_EL3 characteristics are:

Purpose

Holds the faulting Virtual Address for all synchronous Instruction or Data Abort and PC alignment fault exceptions that are taken to EL3.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to FAR_EL3 are UNDEFINED.

Attributes

FAR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Faulting Virtual Address for synchronous exceptions taken to EL3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Faulting Virtual Address for synchronous exceptions taken to EL3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Faulting Virtual Address for synchronous exceptions taken to EL3. Exceptions that set the FAR_EL3 are Instruction Aborts (EC 0x20 or 0x21), Data Aborts (EC 0x24 or 0x25), and PC alignment faults (EC 0x22). [ESR_EL3](#).EC holds the EC value for the exception.

For a synchronous External abort, if the VA that generated the abort was from an address range for which $\text{TCR_ELx.TBI}\{<0|1>\} == 1$ for the translation regime in use when the abort was generated, then the top eight bits of FAR_EL3 are UNKNOWN.

For a synchronous External abort other than a synchronous External abort on a translation table walk, this field is valid only if [ESR_EL3](#).FnV is 0, and the FAR_EL3 is UNKNOWN if [ESR_EL3](#).FnV is 1.

For all other exceptions taken to EL3, the FAR_EL3 is UNKNOWN.

If a memory fault that sets FAR_EL3, other than a Tag Check Fault, is generated from a data cache maintenance or other DC instruction, this field holds the address specified in the register argument of the instruction.

On an exception due to a Tag Check Fault caused by a data cache maintenance or other DC instruction, the address held in FAR_EL3 is IMPLEMENTATION DEFINED as one of the following:

- The lowest address that gave rise to the fault.
- The address specified in the register argument of the instruction as generated by MMU faults caused by [DC ZVA](#).

If the exception that updates FAR_EL3 is taken from an Exception level using AArch32, the top 32 bits are all zero, unless both of the following apply, in which case the top 32 bits of FAR_ELx are 0x00000001:

- The faulting address was generated by a load or store instruction that sequentially incremented from address 0xFFFFFFFF. Such a load or store instruction is CONSTRAINED UNPREDICTABLE.
- The implementation treats such incrementing as setting bit[32] of the virtual address to 1.

For a Data Abort or Watchpoint exception, if address tagging is enabled for the address accessed by the data access that caused the exception, then this field includes the tag. For more information about address tagging, see 'Address tagging in AArch64 state'.

For a synchronous Tag Check Fault abort, bits[63:60] are UNKNOWN.

Execution at EL2, EL1 or EL0 makes FAR_EL3 become UNKNOWN.

Note

The address held in this register is an address accessed by the instruction fetch or data access that caused the exception that actually gave rise to the instruction or data abort. It is the lowest address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores an unaligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

FAR_EL3 is made UNKNOWN on an exception return from EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing FAR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, FAR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0110 | 0b0000 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return FAR_EL3;
```

MSR FAR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0110 | 0b0000 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    FAR_EL3 = X[t];
```

FPCR, Floating-point Control Register

The FPCR characteristics are:

Purpose

Controls floating-point behavior.

Configuration

AArch64 System register FPCR bits [26:15] are architecturally mapped to AArch32 System register [FPSCR\[26:15\]](#).

AArch64 System register FPCR bits [12:8] are architecturally mapped to AArch32 System register [FPSCR\[12:8\]](#).

It is IMPLEMENTATION DEFINED whether the Len and Stride fields can be programmed to non-zero values, which will cause some AArch32 floating-point instruction encodings to be UNDEFINED, or whether these fields are RAZ.

Attributes

FPCR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|--------|----|----|----|------|----|----|----|-----|----|----|----|-----|--|--|--|------|--|--|--|-----|--|--|--|-----|--|--|--|-----|--|--|--|------|--|--|--|-----|--|--|--|-----|--|--|--|------|--|--|--|-----|--|--|--|----|--|--|--|-----|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | AHP | | | | DN | | | | FZ | | | | RMode | | | | Stride | | | | FZ16 | | | | Len | | | | IDE | | | | RES0 | | | | EBF | | | | IXE | | | | UFE | | | | OFED | | | | DZE | | | | IOE | | | | RES0 | | | | NEP | | | | AH | | | | FIZ | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:27]

Reserved, RES0.

AHP, bit [26]

Alternative half-precision control bit.

| AHP | Meaning |
|-----|---|
| 0b0 | IEEE half-precision format selected. |
| 0b1 | Alternative half-precision format selected. |

This bit is used only for conversions between half-precision floating-point and other floating-point formats.

The data-processing instructions added as part of the FEAT_FP16 extension always use the IEEE half-precision format, and ignore the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DN, bit [25]

Default NaN use for NaN propagation.

| DN | Meaning |
|-----|--|
| 0b0 | NaN operands propagate through to the output of a floating-point operation. |
| 0b1 | Any operation involving one or more NaNs returns the Default NaN. This bit has no effect on the output of FABS, FMAX*, FMIN*, and FNEG instructions, and a default NaN is never returned as a result of these instructions. |

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FZ, bit [24]

Flushing denormalized numbers to zero control bit.

| FZ | Meaning |
|-----|--|
| 0b0 | If FPCR.AH is 0, the flushing to zero of single-precision and double-precision denormalized inputs to, and outputs of, floating-point instructions not enabled by this control, but other factors might cause the input denormalized numbers to be flushed to zero. If FPCR.AH is 1, the flushing to zero of single-precision and double-precision denormalized outputs of floating-point instructions not enabled by this control, but other factors might cause the input denormalized numbers to be flushed to zero. |
| 0b1 | If FPCR.AH is 0, denormalized single-precision and double-precision inputs to, and outputs from, floating-point instructions are flushed to zero. If FPCR.AH is 1, denormalized single-precision and double-precision outputs from floating-point instructions are flushed to zero. |

For more information, see 'Flushing denormalized numbers to zero' and the pseudocode of the floating-point instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RMode, bits [23:22]

Rounding Mode control field.

| RMode | Meaning |
|-------|---|
| 0b00 | Round to Nearest (RN) mode. |
| 0b01 | Round towards Plus Infinity (RP) mode. |
| 0b10 | Round towards Minus Infinity (RM) mode. |
| 0b11 | Round towards Zero (RZ) mode. |

The specified rounding mode is used by both scalar and Advanced SIMD floating-point instructions.

If FPCR.AH is 1, then the following instructions use Round to Nearest mode regardless of the value of this bit:

- The FRECPPE, FRECPSP, FRECPXP, FRSQRTE, and FRSQRSTS instructions.
- The BFCVT, BFCVTN, BFCVTN2, BFCVTNT, BFMLALB, and BFMLALT instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Stride, bits [21:20]

This field has no function in AArch64 state, and non-zero values are ignored during execution in AArch64 state.

This field is included only for context saving and restoration of the AArch32 [FPSCR](#).Stride field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FZ16, bit [19]

When FEAT_FP16 is implemented:

Flushing denormalized numbers to zero control bit on half-precision data-processing instructions.

| FZ16 | Meaning |
|------|--|
| 0b0 | For some instructions, this bit disables flushing to zero of inputs and outputs that are half-precision denormalized numbers. |
| 0b1 | Flushing denormalized numbers to zero enabled. For some instructions that do not convert a half-precision input to a higher precision output, this bit enables flushing to zero of inputs and outputs that are half-precision denormalized numbers. |

The value of this bit applies to both scalar and Advanced SIMD floating-point half-precision calculations.

For more information, see 'Flushing denormalized numbers to zero' and the pseudocode of the floating-point instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Len, bits [18:16]

This field has no function in AArch64 state, and non-zero values are ignored during execution in AArch64 state.

This field is included only for context saving and restoration of the AArch32 [FPSCR](#).Len field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IDE, bit [15]

Input Denormal floating-point exception trap enable.

| IDE | Meaning |
|-----|--|
| 0b0 | Untrapped exception handling selected. If the floating-point exception occurs, the FPSR .IDC bit is set to 1. |
| 0b1 | Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR .IDC bit. |

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [14]

Reserved, RES0.

EBF, bit [13]

When FEAT_SME is implemented:

The value of this bit controls the numeric behaviors of BFloat16 dot product calculations performed by the BFDOT, BFMMMLA, BFMOA, and BFMOA instructions.

When [ID_AA64ISAR1_EL1.BF16](#) and [ID_AA64ZFR0_EL1.BF16](#) are 0b0010, the PE supports the FPCR.EBF field. Otherwise, FPCR.EBF is RES0.

| EBF | Meaning |
|-----|--|
| 0b0 | These instructions use the standard BFloat16 behaviors: <ul style="list-style-type: none"> Ignoring the FPCR.RMode control and using the rounding mode defined for BFloat16. For more information, see 'Round to Odd mode'. Flushing denormalized inputs and outputs to zero, as if the FPCR.FZ and FPCR.FIZ controls had the value '1'. Performing unfused multiplies and additions with intermediate rounding of all products and sums. |
| 0b1 | These instructions use the extended BFloat16 behaviors: <ul style="list-style-type: none"> Supporting all four IEEE 754 rounding modes selected by the FPCR.RMode control. Optionally, flushing denormalized inputs and outputs to zero, as governed by the FPCR.FZ and FPCR.FIZ controls. Performing a fused sum-of-products without intermediate rounding of the products, but rounding the single-precision sum before addition to the accumulator. Generating the default NaN as intermediate sum-of-products when any multiplier input is a NaN, or any product is infinity \times 0.0, or there are infinite products with differing signs. Generating an intermediate sum-of-products of the same infinity when there are infinite products all with the same sign. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IXE, bit [12]

Inexact floating-point exception trap enable.

| IXE | Meaning |
|-----|---|
| 0b0 | Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.IXC bit is set to 1. |
| 0b1 | Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.IXC bit. |

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFE, bit [11]

Underflow floating-point exception trap enable.

| UFE | Meaning |
|------------|--|
| 0b0 | Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.UFC bit is set to 1. |
| 0b1 | Trapped exception handling selected. If the floating-point exception occurs and Flush-to-zero is not enabled, the PE does not update the FPSR.UFC bit. |

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFE, bit [10]

Overflow floating-point exception trap enable.

| OFE | Meaning |
|------------|---|
| 0b0 | Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.OFC bit is set to 1. |
| 0b1 | Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.OFC bit. |

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZE, bit [9]

Divide by Zero floating-point exception trap enable.

| DZE | Meaning |
|------------|---|
| 0b0 | Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.DZC bit is set to 1. |
| 0b1 | Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.DZC bit. |

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOE, bit [8]

Invalid Operation floating-point exception trap enable.

| IOE | Meaning |
|------------|---|
| 0b0 | Untrapped exception handling selected. If the floating-point exception occurs, the FPSR.IOC bit is set to 1. |
| 0b1 | Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the FPSR.IOC bit. |

The value of this bit controls both scalar and Advanced SIMD floating-point arithmetic.

If the implementation does not support this exception, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:3]

Reserved, RES0.

NEP, bit [2]

When FEAT_AFP is implemented:

Controls how the output elements other than the lowest element of the vector are determined for Advanced SIMD scalar instructions.

| NEP | Meaning |
|-----|--|
| 0b0 | Does not affect how the output elements other than the lowest are determined for Advanced SIMD scalar instructions. |
| 0b1 | <p>The output elements other than the lowest are taken from the following registers:</p> <ul style="list-style-type: none"> • For 3-input scalar versions of the FMLA (by element) and FMLS (by element) instructions, the <Hd>, <Sd>, or <Dd> register. • For 3-input versions of the FMADD, FMSUB, FNMADD, and FNMSUB instructions, the <Ha>, <Sa>, or <Da> register. • For 2-input scalar versions of the FACGE, FACGT, FCMEQ (register), FCMGE (register), and FCMGT (register) instructions, the <Hm>, <Sm>, or <Dm> register. • For 2-input scalar versions of the FABD, FADD (scalar), FDIV (scalar), FMAX (scalar), FMAXNM (scalar), FMIN (scalar), FMINNM (scalar), FMUL (by element), FMUL (scalar), FMULX (by element), FMULX, FNMUL (scalar), FRECPs, FRSQRTS, and FSUB (scalar) instructions, the <Hn>, <Sn>, or <Dn> register. • For 1-input scalar versions of the following instructions, the <Hd>, <Sd>, or <Dd> register: <ul style="list-style-type: none"> ◦ The (vector) versions of the FCVTAS, FCVTAU, FCVTMS, FCVTMU, FCVTNS, FCVTNU, FCVTPS, and FCVTPU instructions. ◦ The (vector, fixed-point) and (vector, integer) versions of the FCVTZS, FCVTZU, SCVTF, and UCVTF instructions. ◦ The (scalar) versions of the FABS, FNEG, FRINT32X, FRINT32Z, FRINT64X, FRINT64Z, FRINTA, FRINTI, FRINTM, FRINTN, FRINTP, FRINTX, FRINTZ, and FSQRT instructions. ◦ The (scalar, fixed-point) and (scalar, integer) versions of the SCVTF and UCVTF instructions. ◦ The BFCVT, FCVT, FCVTXN, FRECPE, FRECPX, and FRSQRTE instructions. |

The value of FPCR.NEP is treated as 0 for all purposes other than a direct read or write of the FPCR when the PE is in Streaming SVE mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AH, bit [1]

When FEAT_AFP is implemented:

Alternate Handling. Controls alternate handling of floating-point numbers.

The Arm architecture supports two models for handling some of the corner cases of the floating-point behaviors, such as the nature of flushing of denormalized numbers, the detection of tininess and other exceptions and a range of other behaviors. The value of the FPCR.AH bit selects between these models.

For more information on the FPCR.AH bit, see 'Flushing denormalized numbers to zero', 'Floating-point exceptions and exception traps' and the pseudocode of the floating-point instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FIZ, bit [0]

When FEAT_AFP is implemented:

Flush Inputs to Zero. Controls whether single-precision, double-precision and BFloat16 input operands that are denormalized numbers are flushed to zero.

| FIZ | Meaning |
|-----|---|
| 0b0 | The flushing to zero of single-precision and double-precision denormalized inputs to floating-point instructions not enabled by this control, but other factors might cause the input denormalized numbers to be flushed to zero. |
| 0b1 | Denormalized single-precision and double-precision inputs to most floating-point instructions flushed to zero. |

For more information, see 'Flushing denormalized numbers to zero' and the pseudocode of the floating-point instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing FPCR

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, FPCR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CPACR_EL1.FPEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x00);
        else
            AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.FPEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        return FPCR;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif CPACR_EL1.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        return FPCR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        return FPCR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        return FPCR;

```

MSR FPCR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CPACR_EL1.FPEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x00);
        else
            AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.FPEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPCR = X[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif CPACR_EL1.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPCR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPCR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPCR = X[t];

```

FPEXC32_EL2, Floating-Point Exception Control register

The FPEXC32_EL2 characteristics are:

Purpose

Allows access to the AArch32 register [FPEXC](#) from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

AArch64 System register FPEXC32_EL2 bits [31:0] are architecturally mapped to AArch32 System register [FPEXC\[31:0\]](#).

This register is present only when EL1 is capable of using AArch32. Otherwise, direct accesses to FPEXC32_EL2 are UNDEFINED.

If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not RES0.

Implemented only if the implementation includes the Advanced SIMD and floating-point functionality.

Attributes

FPEXC32_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|-----|------|----|-----|------|----|----|----|----|----|----|----|----|----|--------|----|----|-----|------|-------|-----|-----|-----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| EX | EN | DEX | FP2V | VV | TFV | RES0 | | | | | | | | | | VECITR | | | IDF | RES0 | XFUFF | OFF | DZF | IOF | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

EX, bit [31]

Exception bit. From Armv8, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EN, bit [30]

Enables access to the Advanced SIMD and floating-point functionality from all Exception levels, except that setting this field to 0 does not disable the following:

- VMSR accesses to the [FPEXC](#) or [FPSID](#).
- VMRS accesses from the [FPEXC](#), [FPSID](#), [MVFR0](#), [MVFR1](#), or [MVFR2](#).

| EN | Meaning |
|-----|--|
| 0b0 | Accesses to the FPSCR , and any of the SIMD and floating-point registers Q0-Q15, including their views as D0-D31 registers or S0-S31 registers, are UNDEFINED at all Exception levels. |
| 0b1 | This control permits access to the Advanced SIMD and floating-point functionality at all Exception levels. |

Execution of Advanced SIMD and floating-point instructions in AArch32 state can be disabled or trapped by the following controls:

- [CPACR](#).cp10, or, if executing at EL0, [CPACR_EL1](#).FPEN.
- FPEXC.EN.
- If executing in Non-secure state:
 - [HCPTR](#).TCP10, or if EL2 is using AArch64, [CPTR_EL2](#).TFP.
 - [NSACR](#).cp10, or if EL3 is using AArch64, [CPTR_EL3](#).TFP.
- For Advanced SIMD instructions only:
 - [CPACR](#).ASEDIS.
 - If executing in Non-secure state, [HCPTR](#).TASE and [NSACR](#).NSTRCDIS.

See the descriptions of the controls for more information.

Note

When executing at EL0 using AArch32:

- If EL1 is using AArch64, then the Effective value of [FPEXC](#).EN is 1.
- If EL2 is using AArch64 and is enabled in the current Security state, [HCR_EL2](#).TGE is 1, and the Effective value of [HCR_EL2](#).RW is 1, then the Effective value of [FPEXC](#).EN is 1. However, Arm deprecates using the value of FPEXC32_EL2.EN to determine behavior.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DEX, bit [29]

Defined synchronous exception on floating-point execution.

This field identifies whether a synchronous exception generated by the attempted execution of an instruction was generated by an unallocated encoding. The instruction must be in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr() returning TRUE. This field also indicates whether the FPEXC32_EL2.TFV field is valid.

The meaning of this bit is:

| DEX | Meaning |
|-----|---|
| 0b0 | The exception was generated by the attempted execution of an unallocated instruction in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr(). If FPEXC32_EL2.TFV is RW then it is invalid and UNKNOWN. If FPEXC32_EL2.{IDF, IXF, UFF, OFF, DZF, IOF} are RW then they are invalid and UNKNOWN. |
| 0b1 | The exception was generated during the execution of an allocated encoding. FPEXC32_EL2.TFV is valid and indicates the cause of the exception. |

On an exception that sets this bit to 1 the exception-handling routine must clear this bit to 0.

On an implementation that both does not support trapping of floating-point exceptions and implements the AArch32 [FPSCR](#).{Stride, Len} fields as RAZ, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FP2V, bit [28]

FPINST2 instruction valid bit. From Armv8, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VV, bit [27]

VECITR valid bit. From Armv8, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TFV, bit [26]

Trapped Fault Valid bit. Valid only when the value of FPEXC.DEX is 1. When valid, it indicates the cause of the exception and therefore whether the FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} bits are valid.

| TFV | Meaning |
|-----|---|
| 0b0 | The exception was caused by the execution of a floating-point VABS, VADD, VDIV, VFMA, VFMS, VFNMA, VFNMS, VMLA, VMLS, VMOV, VMUL, VNEG, VNMLA, VNMLS, VNMUL, VSQRT, or VSUB instruction when one or both of FPSCR .{Stride, Len} was non-zero. If the FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} bits are RW then they are invalid and UNKNOWN. |
| 0b1 | FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} indicate the presence of trapped floating-point exceptions that had occurred at the time of the exception. Bits are set for all trapped exceptions that had occurred at the time of the exception. |

This bit returns a status value and ignores writes.

When the value of FPEXC.DEX is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

On an implementation that supports the trapping of floating-point exceptions and implements [FPSCR](#).{Stride, Len} as RAZ, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [25:11]

Reserved, RES0.

VECITR, bits [10:8]

Vector iteration count. From Armv8, this field is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Input Denormal exception occurred while [FPSCR](#).IDE was 1:

| IDF | Meaning |
|-----|--|
| 0b0 | Input Denormal exception has not occurred. |
| 0b1 | Input Denormal exception has occurred. |

Input Denormal exceptions can occur only when [FPSCR.FZ](#) is 1.

Note

A half-precision floating-point value that is flushed to zero because the value of [FPSCR.FZ16](#) is 1 does not generate an Input Denormal exception.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC32_EL2.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Inexact exception occurred while [FPSCR.IXE](#) was 1:

| IXF | Meaning |
|-----|-------------------------------------|
| 0b0 | Inexact exception has not occurred. |
| 0b1 | Inexact exception has occurred. |

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFF, bit [3]

Underflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Underflow exception occurred while [FPSCR.UFE](#) was 1:

| UFF | Meaning |
|-----|---------------------------------------|
| 0b0 | Underflow exception has not occurred. |
| 0b1 | Underflow exception has occurred. |

Underflow trapped exceptions can occur:

- On half-precision data-processing instructions only when [FPSCR.FZ16](#) is 0.
- Otherwise only when [FPSCR.FZ](#) is 0.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC32_EL2.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFF, bit [2]

Overflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Overflow exception occurred while [FPSCR.OFE](#) was 1:

| OFF | Meaning |
|-----|--------------------------------------|
| 0b0 | Overflow exception has not occurred. |
| 0b1 | Overflow exception has occurred. |

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZF, bit [1]

Divide by Zero trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether a Divide by Zero exception occurred while [FPSCR.DZE](#) was 1:

| DZF | Meaning |
|-----|--|
| 0b0 | Divide by Zero exception has not occurred. |
| 0b1 | Divide by Zero exception has occurred. |

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOF, bit [0]

Invalid Operation trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Invalid Operation exception occurred while [FPSCR.IOE](#) was 1:

| IOF | Meaning |
|-----|---|
| 0b0 | Invalid Operation exception has not occurred. |
| 0b1 | Invalid Operation exception has occurred. |

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing FPEXC32_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, FPEXC32_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        return FPEXC32_EL2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        return FPEXC32_EL2;

```

MSR FPEXC32_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPEXC32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPEXC32_EL2 = X[t];

```


FPSR, Floating-point Status Register

The FPSR characteristics are:

Purpose

Provides floating-point system status information.

Configuration

AArch64 System register FPSR bits [31:27] are architecturally mapped to AArch32 System register [FPSCR\[31:27\]](#).

AArch64 System register FPSR bit [7] is architecturally mapped to AArch32 System register [FPSCR\[7\]](#).

AArch64 System register FPSR bits [4:0] are architecturally mapped to AArch32 System register [FPSCR\[4:0\]](#).

Attributes

FPSR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|------|-----|-----|-----|-----|-----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | Z | C | V | QC | RES0 | | | | | | | | | | | | | | | | | IDC | RES0 | IXC | UFC | OFD | DZC | IOC | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

N, bit [31]

When AArch32 is supported and AArch32 floating-point is implemented:

Negative condition flag for AArch32 floating-point comparison operations.

Note

AArch64 floating-point comparisons set the PSTATE.N flag instead.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Z, bit [30]

When AArch32 is supported and AArch32 floating-point is implemented:

Zero condition flag for AArch32 floating-point comparison operations.

Note

AArch64 floating-point comparisons set the PSTATE.Z flag instead.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

C, bit [29]

When AArch32 is supported and AArch32 floating-point is implemented:

Carry condition flag for AArch32 floating-point comparison operations.

Note

AArch64 floating-point comparisons set the PSTATE.C flag instead.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

V, bit [28]

When AArch32 is supported and AArch32 floating-point is implemented:

Overflow condition flag for AArch32 floating-point comparison operations.

Note

AArch64 floating-point comparisons set the PSTATE.V flag instead.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

QC, bit [27]

Cumulative saturation bit, Advanced SIMD only. This bit is set to 1 to indicate that an Advanced SIMD integer operation has saturated since 0 was last written to this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [26:8]

Reserved, RES0.

IDC, bit [7]

Input Denormal cumulative floating-point exception bit. This bit is set to 1 to indicate that the Input Denormal floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the [FPCR.IDE](#) bit. This bit is set to 1 to indicate a floating-point exception only if [FPCR.IDE](#) is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXC, bit [4]

Inexact cumulative floating-point exception bit. This bit is set to 1 to indicate that the Inexact floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the [FPCR.IXE](#) bit. This bit is set to 1 to indicate a floating-point exception only if [FPCR.IXE](#) is 0.

The criteria for the Inexact floating-point exception to occur are affected by whether denormalized numbers are flushed to zero and by the value of the [FPCR.AH](#) bit. For more information, see 'Floating-point exceptions and exception traps'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFC, bit [3]

Underflow cumulative floating-point exception bit. This bit is set to 1 to indicate that the Underflow floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the [FPCR.UFE](#) bit. This bit is set to 1 to indicate a floating-point exception only if [FPCR.UFE](#) is 0 or if flushing denormalized numbers to zero is enabled.

The criteria for the Underflow floating-point exception to occur are affected by whether denormalized numbers are flushed to zero and by the value of the [FPCR.AH](#) bit. For more information, see 'Floating-point exceptions and exception traps'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFC, bit [2]

Overflow cumulative floating-point exception bit. This bit is set to 1 to indicate that the Overflow floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the [FPCR.OFE](#) bit. This bit is set to 1 to indicate a floating-point exception only if [FPCR.OFE](#) is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZC, bit [1]

Divide by Zero cumulative floating-point exception bit. This bit is set to 1 to indicate that the Divide by Zero floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the [FPCR.DZE](#) bit. This bit is set to 1 to indicate a floating-point exception only if [FPCR.DZE](#) is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOC, bit [0]

Invalid Operation cumulative floating-point exception bit. This bit is set to 1 to indicate that the Invalid Operation floating-point exception has occurred since 0 was last written to this bit.

How scalar and Advanced SIMD floating-point instructions update this bit depends on the value of the [FPCR.IOE](#) bit. This bit is set to 1 to indicate a floating-point exception only if [FPCR.IOE](#) is 0.

The criteria for the Invalid Operation floating-point exception to occur are affected by the value of the [FPCR.AH](#) bit. For more information, see 'Floating-point exceptions and exception traps'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing FPSR

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, FPSR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CPACR_EL1.FPEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x00);
        else
            AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.FPEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        return FPSR;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif CPACR_EL1.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        return FPSR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        return FPSR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        return FPSR;

```

MSR FPSR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CPACR_EL1.FPEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x00);
        else
            AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.FPEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPSR = X[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif CPACR_EL1.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPSR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TFP == '1' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x07);
    elsif HaveEL(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPSR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TFP == '1' then
        AArch64.SystemAccessTrap(EL3, 0x07);
    else
        FPSR = X[t];

```

GCR_EL1, Tag Control Register.

The GCR_EL1 characteristics are:

Purpose

Tag Control Register.

Configuration

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to GCR_EL1 are UNDEFINED.

Attributes

GCR_EL1 is a 64-bit register.

Field descriptions

Diagram illustrating the bit layout of a 64-bit register:

- Bits 63 to 16 are labeled **RES0** (Reserved).
- Bits 15 to 14 are labeled **RRND** (Round/Remainder).
- Bits 13 to 0 are labeled **Exclude** (Excluded).

Bits [63:17]

Reserved, RES0.

RRND, bit [16]

Controls generation of tag values by the IRG instruction.

| RRND | Meaning |
|------|--|
| 0b0 | IRG generates a tag value as defined by RandomTag(). |
| 0b1 | IRG generates an implementation-specific tag value with a distribution of tag values no worse than generated with GCR_EL1.RRND == 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Exclude, bits [15:0]

Allocation Tag values excluded from selection by ChooseNonExcludedTag().

If all bits of GCR_EL1.Exclude are 1, then the Allocation Tag value 0 will be used.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, GCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return GCR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return GCR_EL1;
elsif PSTATE.EL == EL3 then
    return GCR_EL1;

```

MSR GCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        GCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    GCR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GMID_EL1, Multiple tag transfer ID register

The GMID_EL1 characteristics are:

Purpose

Indicates the block size that is accessed by the LDGM and STGM System instructions.

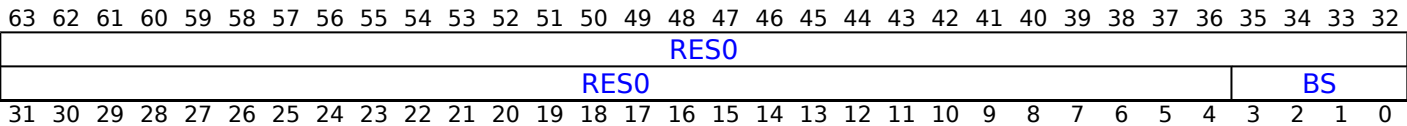
Configuration

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to GMID_EL1 are UNDEFINED.

Attributes

GMID_EL1 is a 64-bit register.

Field descriptions



Bits [63:4]

Reserved, RES0.

BS, bits [3:0]

Log₂ of the block size in words. The minimum supported size is 16B (value == 2) and the maximum is 256B (value == 6).

Accessing GMID_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, GMID_EL1

| CRn | op0 | op1 | op2 | CRm |
|--------|------|-------|-------|--------|
| 0b0000 | 0b11 | 0b001 | 0b100 | 0b0000 |

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID5 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return GMID_EL1;
elseif PSTATE.EL == EL2 then
    return GMID_EL1;
elseif PSTATE.EL == EL3 then
    return GMID_EL1;
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GPCCR_EL3, Granule Protection Check Control Register (EL3)

The GPCCR_EL3 characteristics are:

Purpose

The control register for Granule Protection Checks.

Configuration

This register is present only when FEAT_RME is implemented. Otherwise, direct accesses to GPCCR_EL3 are UNDEFINED.

Attributes

GPCCR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|---------|----|----|----|------|----|------|-----|-----|----|------|------|------|----|----|----|----|----|----|----|-----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | LOGPTSZ | | | | RES0 | | GPCP | GPC | PGS | SH | ORGN | IRGN | RES0 | | | | | | | | PPS | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

LOGPTSZ, bits [23:20]

Level 0 GPT entry size.

This field advertises the number of least-significant address bits protected by each entry in the level 0 GPT.

| LOGPTSZ | Meaning |
|---------|--|
| 0b0000 | 30-bits. Each entry covers 1GB of address space. |
| 0b0100 | 34-bits. Each entry covers 16GB of address space. |
| 0b0110 | 36-bits. Each entry covers 64GB of address space. |
| 0b1001 | 39-bits. Each entry covers 512GB of address space. |

All other values are reserved.

Access to this field is **RO**.

Bits [19:18]

Reserved, RES0.

GPCP, bit [17]

Granule Protection Check Priority.

This control governs behavior of granule protection checks on fetches of stage 2 Table descriptors.

| GPCP | Meaning |
|------|--|
| 0b0 | GPC faults are all reported with a priority that is consistent with the GPC being performed on any access to physical address space. |
| 0b1 | A GPC fault for the fetch of a Table descriptor for a stage 2 translation table walk might not be generated or reported. All other GPC faults are reported with a priority consistent with the GPC being performed on all accesses to physical address spaces. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GPC, bit [16]

Granule Protection Check Enable.

| GPC | Meaning |
|-----|--|
| 0b0 | Granule protection checks are disabled. Accesses are not prevented by this mechanism. |
| 0b1 | All accesses to physical address spaces are subject to granule protection checks, except for fetches of GPT information and accesses governed by the GPCCR_EL3.GPCP control. |

If any stage of translation is enabled, this bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

PGS, bits [15:14]

Physical Granule size.

| PGS | Meaning |
|------|---------|
| 0b00 | 4KB. |
| 0b01 | 64KB. |
| 0b10 | 16KB. |

All other values are reserved.

The value of this field is permitted to be cached in a TLB.

Granule sizes not supported for stage 1 and not supported for stage 2, as defined in [ID_AA64MMFR0_EL1](#), are reserved. For example, if [ID_AA64MMFR0_EL1.TGran16](#) == 0b0000 and [ID_AA64MMFR0_EL1.TGran16_2](#) == 0b0001, then the PGS encoding 0b10 is reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH, bits [13:12]

GPT fetch Shareability attribute

| SH | Meaning |
|------|------------------|
| 0b00 | Non-shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

All other values are reserved.

Fetches of GPT information are made with the Shareability attribute that is configured in this field.

If both ORGN and IRGN are configured with Non-cacheable attributes, it is invalid to configure this field to any value other than 0b10.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN, bits [11:10]

GPT fetch Outer cacheability attribute.

| ORGN | Meaning |
|------|---|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable. |

Fetches of GPT information are made with the Outer cacheability attributes configured in this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN, bits [9:8]

GPT fetch Inner cacheability attribute.

| IRGN | Meaning |
|------|---|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable. |

Fetches of GPT information are made with the Inner cacheability attributes configured in this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:3]

Reserved, RES0.

PPS, bits [2:0]

Protected Physical Address Size.

The size of the memory region protected by [GPTBR_EL3](#), in terms of the number of least-significant address bits.

| PPS | Meaning |
|-------|---|
| 0b000 | 32 bits, 4GB protected address space. |
| 0b001 | 36 bits, 64GB protected address space. |
| 0b010 | 40 bits, 1TB protected address space. |
| 0b011 | 42 bits, 4TB protected address space. |
| 0b100 | 44 bits, 16TB protected address space. |
| 0b101 | 48 bits, 256TB protected address space. |
| 0b110 | 52 bits, 4PB protected address space. |

All other values are reserved.

Configuration of this field to a value exceeding the implemented physical address size is invalid.

The value of this field is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GPCCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, GPCCR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0010 | 0b0001 | 0b110 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return GPCCR_EL3;
```

MSR GPCCR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0010 | 0b0001 | 0b110 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    GPCCR_EL3 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GPTBR_EL3, Granule Protection Table Base Register

The GPTBR_EL3 characteristics are:

Purpose

The control register for Granule Protection Table base address.

Configuration

This register is present only when FEAT_RME is implemented. Otherwise, direct accesses to GPTBR_EL3 are UNDEFINED.

Attributes

GPTBR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | BADDR | | | | | | | | | | | |
| BADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

Bits [63:40]

Reserved, RES0.

BADDR, bits [39:0]

Base address for the level 0 GPT.

This field represents bits [51:12] of the level 0 GPT base address.

The level 0 GPT is aligned in memory to the greater of:

- The size of the level 0 GPT in bytes.
- 4KB.

Bits [x:0] of the base address are treated as zero, where:

- $x = \text{Max}(\text{pps} - \text{lgptsz} + 2, 11)$
- pps is derived from [GPCCR_EL3.PPS](#) as follows:

| GPCCR_EL3.PPS | pps |
|---------------|-----|
| 0b000 | 32 |
| 0b001 | 36 |
| 0b010 | 40 |
| 0b011 | 42 |
| 0b100 | 44 |
| 0b101 | 48 |
| 0b110 | 52 |

- lgptsz is derived from [GPCCR_EL3.LOGPTSZ](#) as follows:

| GPCCR_EL3.LOGPTSZ | lgptsz |
|-------------------|--------|
| 0b0000 | 30 |
| 0b0100 | 34 |
| 0b0110 | 36 |
| 0b1001 | 39 |

If x is greater than 11, then BADDR[$x - 12:0$] are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GPTBR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, GPTBR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0010 | 0b0001 | 0b100 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return GPTBR_EL3;
```

MSR GPTBR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0010 | 0b0001 | 0b100 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    GPTBR_EL3 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HACR_EL2, Hypervisor Auxiliary Control Register

The HACR_EL2 characteristics are:

Purpose

Controls trapping to EL2 of IMPLEMENTATION DEFINED aspects of EL1 or EL0 operation.

Note

Arm recommends that the values in this register do not cause unnecessary traps to EL2 when [HCR_EL2](#).{E2H, TGE} == {1, 1}.

Configuration

AArch64 System register HACR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HACR\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

HACR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HACR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HACR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    return HACR_EL2;
elseif PSTATE.EL == EL3 then
    return HACR_EL2;

```

MSR HACR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    HACR_EL2 = X[t];
elseif PSTATE.EL == EL3 then
    HACR_EL2 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HAFGRTR_EL2, Hypervisor Activity Monitors Fine-Grained Read Trap Register

The HAFGRTR_EL2 characteristics are:

Purpose

Provides controls for traps of MRS reads of Activity Monitors System registers.

Configuration

This register is present only when FEAT_AMUv1 is implemented and FEAT_FGT is implemented. Otherwise, direct accesses to HAFGRTR_EL2 are UNDEFINED.

Attributes

HAFGRTR_EL2 is a 64-bit register.

Field descriptions

| | | | | | |
|-----------------|----------------|-----------------|----------------|-----------------|----------------|
| 63 | 62 | 61 | 60 | 59 | 58 |
| AMEVTYPER16_EL0 | AMEVCNTR16_EL0 | AMEVTYPER15_EL0 | AMEVCNTR15_EL0 | AMEVTYPER14_EL0 | AMEVCNTR14_EL0 |
| 31 | 30 | 29 | 28 | 27 | 26 |

Bits [63:50]

Reserved, RES0.

AMEVTYPER1<x>_EL0, bit [19+2x], for x = 15 to 0
When AMEVTYPER1<x> is implemented:

Trap MRS reads of AMEVTYPER1<x>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<x> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| AMEVTYPER1<x>_EL0 | Meaning |
|-------------------|--|
| 0b0 | MRS reads of AMEVTYPER1<x>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVTYPER1<x> at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none">MRS reads of AMEVTYPER1<x>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18.MRC reads of AMEVTYPER1<x> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

AMEVCNTR1<x>_EL0, bit [18+2x], for x = 15 to 0

When AMEVCNTR1<x> is implemented:

Trap MRS reads of [AMEVCNTR1<x>_EL0](#) at EL1 and EL0 using AArch64 and MRC reads of [AMEVCNTR1<x>](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| AMEVCNTR1<x>_EL0 | Meaning |
|------------------|---|
| 0b0 | MRS reads of AMEVCNTR1<x>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR1<x> at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> MRS reads of AMEVCNTR1<x>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MRC reads of AMEVCNTR1<x> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

AMCNTEN<x>, bit [17x], for x = 1 to 0

Trap MRS reads and MRC reads of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MRS reads of [AMCNTENCLR<x>_EL0](#) and [AMCNTENSET<x>_EL0](#).
- At EL0 using AArch32 when EL1 is using AArch64: MRC reads of [AMCNTENCLR<x>](#) and [AMCNTENSET<x>](#).

| AMCNTEN<x> | Meaning |
|------------|---|
| 0b0 | The operations listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> MRS reads at EL1 and EL0 using AArch64 of AMCNTENCLR<x>_EL0 and AMCNTENSET<x>_EL0 are trapped to EL2 and reported with EC syndrome value 0x18. MRC reads at EL0 using AArch32 of AMCNTENCLR<x> and AMCNTENSET<x> are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Bits [16:5]

Reserved, RES0.

AMEVCNTR0<x>_EL0, bit [x+1], for x = 3 to 0

Trap MRS reads of [AMEVCNTR0<x>_EL0](#) at EL1 and EL0 using AArch64 and MRC reads of [AMEVCNTR0<x>](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| AMEVCNTR0<x>_EL0 | Meaning |
|------------------|---|
| 0b0 | MRS reads of AMEVCNTR0<x>_EL0 at EL1 and EL0 using AArch64 and MRC reads of AMEVCNTR0<x> at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> MRS reads of AMEVCNTR0<x>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MRC reads of AMEVCNTR0<x> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Accessing HAFGRTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HAFGRTR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0011 | 0b0001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x1E8];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return HAFGRTR_EL2;
elsif PSTATE.EL == EL3 then
    return HAFGRTR_EL2;

```

MSR HAFGRTR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0011 | 0b0001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x1E8] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HAFGRTR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    HAFGRTR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HCR_EL2, Hypervisor Configuration Register

The HCR_EL2 characteristics are:

Purpose

Provides configuration controls for virtualization, including defining whether various operations are trapped to EL2.

Configuration

AArch64 System register HCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HCR\[31:0\]](#).

AArch64 System register HCR_EL2 bits [63:32] are architecturally mapped to AArch32 System register [HCR2\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

The bits in this register behave as if they are 0 for all purposes other than direct reads of the register if EL2 is not enabled in the current Security state.

Attributes

HCR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | |
|--------|------|-----|-----|--------|------|------|-----|---------|--------|--------|-------|-----|-------|-------|------|------|------|-----|-----|-----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 |
| TWEDEL | | | | TWEDEn | TID5 | DCT | ATA | TTLBOS | TTLBIS | EnSCXT | TOCU | AMV | OFFEN | TICAB | TID4 | GPF | FIEN | FWB | NV2 | ATN |
| RW | TRVM | HCD | TDZ | TGE | TVM | TTLB | TPU | Bit[23] | TSW | TACR | TIDCP | TSC | TID3 | TID2 | TID1 | TID0 | TWE | TWI | DC | I |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 |

TWEDEL, bits [63:60]
When FEAT_TWED is implemented:

TWE Delay. A 4-bit unsigned number that, when HCR_EL2.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE* caused by HCR_EL2.TWE as $2^{(TWEDEL + 8)}$ cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [59]
When FEAT_TWED is implemented:

TWE Delay Enable. Enables a configurable delayed trap of the WFE* instruction caused by HCR_EL2.TWE.

| TWEDEn | Meaning |
|--------|---|
| 0b0 | The delay for taking the trap is IMPLEMENTATION DEFINED. |
| 0b1 | The delay for taking the trap is at least the number of cycles defined in HCR_EL2.TWEDEL. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TID5, bit [58]

When FEAT_MTE2 is implemented:

Trap ID group 5. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:

AArch64:

- [GMID_EL1](#).

| TID5 | Meaning |
|------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | The specified EL1 and EL0 accesses to ID group 5 registers are trapped to EL2. |

When the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field has an Effective value of 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DCT, bit [57]

When FEAT_MTE2 is implemented:

Default Cacheability Tagging. When HCR_EL2.DC is in effect, controls whether stage 1 translations are treated as Tagged or Untagged.

| DCT | Meaning |
|-----|---|
| 0b0 | Stage 1 translations are treated as Untagged. |
| 0b1 | Stage 1 translations are treated as Tagged. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA, bit [56]

When FEAT_MTE2 is implemented:

Allocation Tag Access. When HCR_EL2.{E2H,TGE} != {1,1}, controls EL1 and EL0 access to Allocation Tags.

| ATA | Meaning |
|-----|--|
| 0b0 | Access to Allocation Tags is prevented. Accesses at EL1 to GCR_EL1 , RGSRR_EL1 , TFSRR_EL1 , TFSRR_EL2 , or TFSRRE0_EL1 that are not UNDEFINED are trapped to EL2. |
| 0b1 | This control does not prevent access to Allocation Tags. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TTLBOS, bit [55]**When FEAT_EVT is implemented:**

Trap TLB maintenance instructions that operate on the Outer Shareable domain. Traps execution of those TLB maintenance instructions at EL1 to EL2, when EL2 is enabled in the current Security state. This applies to the following instructions:

[TLBI VMALLE1OS](#), [TLBI VAE1OS](#), [TLBI ASIDE1OS](#), [TLBI VAAE1OS](#), [TLBI VALE1OS](#), [TLBI VAALE1OS](#), [TLBI RVAE1OS](#), [TLBI RVAAE1OS](#), [TLBI RVALE1OS](#), and [TLBI RVAALE1OS](#).

| TTLBOS | Meaning |
|---------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Execution of the specified instructions are trapped to EL2. |

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TTLBIS, bit [54]**When FEAT_EVT is implemented:**

Trap TLB maintenance instructions that operate on the Inner Shareable domain. Traps execution of those TLB maintenance instructions at EL1 to EL2, when EL2 is enabled in the current Security state. This applies to the following instructions:

- When EL1 is using AArch64, [TLBI VMALLE1IS](#), [TLBI VAE1IS](#), [TLBI ASIDE1IS](#), [TLBI VAAE1IS](#), [TLBI VALE1IS](#), [TLBI VAALE1IS](#), [TLBI RVAE1IS](#), [TLBI RVAAE1IS](#), [TLBI RVALE1IS](#), and [TLBI RVAALE1IS](#).
- When EL1 is using AArch32, [TLBIALLIS](#), [TLBIMVAIS](#), [TLBIASIDIS](#), [TLBIMVAAIS](#), [TLBIMVALIS](#), and [TLBIMVAALIS](#).

| TTLBIS | Meaning |
|---------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Execution of the specified instructions are trapped to EL2. |

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnSCXT, bit [53]**When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:**

Enable Access to the [SCXTNUM_EL1](#) and [SCXTNUM_ELO](#) registers. The defined values are:

| EnSCXT | Meaning |
|--------|--|
| 0b0 | When HCR_EL2.E2H is 0 or HCR_EL2.TGE is 0, and EL2 is enabled in the current Security state, EL1 and EL0 access to SCXTNUM_EL0 and EL1 access to SCXTNUM_EL1 is disabled by this mechanism, causing an exception to EL2, and the values of these registers to be treated as 0. When HCR_EL2.{E2H, TGE} is {1, 1} and EL2 is enabled in the current Security state, EL0 access to SCXTNUM_EL0 is disabled by this mechanism, causing an exception to EL2, and the value of this register to be treated as 0. |
| 0b1 | This control does not cause accesses to SCXTNUM_EL0 or SCXTNUM_EL1 to be trapped. |

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TOCU, bit [52]

When FEAT_EVT is implemented:

Trap cache maintenance instructions that operate to the Point of Unification. Traps execution of those cache maintenance instructions to EL2, when EL2 is enabled in the current Security state. This applies to the following instructions:

- When [SCTLR_EL1](#).UCI is 1, HCR_EL2.{TGE, E2H} is not {1, 1}, and EL0 is using AArch64, [IC IVAU](#), [DC CVAU](#).
- When EL1 is using AArch64, [IC IVAU](#), [IC IALLU](#), [DC CVAU](#).
- When EL1 is using AArch32, [ICIMVAU](#), [IC IALLU](#), [DCCMVAU](#).

Note

An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2. In addition:

- [IC IALLUIS](#) and [IC IALLU](#) are always UNDEFINED at EL0 using AArch64.
- [ICIMVAU](#), [IC IALLU](#), [IC IALLUIS](#), and [DCCMVAU](#) are always UNDEFINED at EL0 using AArch32.

| TOCU | Meaning |
|------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Execution of the specified instructions are trapped to EL2. |

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMVOFFEN, bit [51]

When FEAT_AMUv1p1 is implemented:

Activity Monitors Virtual Offsets Enable.

| AMVOFFEN | Meaning |
|----------|--|
| 0b0 | Virtualization of the Activity Monitors is disabled. Indirect reads of the virtual offset registers are zero. |
| 0b1 | Virtualization of the Activity Monitors is enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TICAB, bit [50]

When FEAT_EVT is implemented:

Trap ICIALLUIS/IC IALLUIS cache maintenance instructions. Traps execution of those cache maintenance instructions at EL1 to EL2, when EL2 is enabled in the current Security state. This applies to the following instructions:

- When EL1 is using AArch64, [IC IALLUIS](#).
- When EL1 is using AArch32, [ICIALLUIS](#).

| TICAB | Meaning |
|-------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 execution of the specified instructions is trapped to EL2. |

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TID4, bit [49]

When FEAT_EVT is implemented:

Trap ID group 4. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:

AArch64:

- EL1 reads of [CCSIDR_EL1](#), [CCSIDR2_EL1](#), [CLIDR_EL1](#), and [CSSELR_EL1](#).
- EL1 writes to [CSSELR_EL1](#).

AArch32:

- EL1 reads of [CCSIDR](#), [CCSIDR2](#), [CLIDR](#), and [CSSELR](#).
- EL1 writes to [CSSELR](#).

| TID4 | Meaning |
|------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | The specified EL1 and EL0 accesses to ID group 4 registers are trapped to EL2. |

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

GPF, bit [48]

When FEAT_RME is implemented:

Controls the reporting of Granule protection faults at EL0 and EL1.

| GPF | Meaning |
|-----|--|
| 0b0 | This control does not cause exceptions to be routed from EL0 and EL1 to EL2. |
| 0b1 | Instruction Aborts and Data Aborts due to GPFs from EL0 and EL1 are routed to EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FIEN, bit [47]

When FEAT_RASv1p1 is implemented:

Fault Injection Enable. Unless this bit is set to 1, accesses to the [ERXPFPCDN_EL1](#), [ERXPFPGCTL_EL1](#), and [ERXPFPGF_EL1](#) registers from EL1 generate a Trap exception to EL2, when EL2 is enabled in the current Security state, reported using EC syndrome value 0x18.

| FIEN | Meaning |
|------|---|
| 0b0 | Accesses to the specified registers from EL1 are trapped to EL2, when EL2 is enabled in the current Security state. |
| 0b1 | This control does not cause any instructions to be trapped. |

If EL2 is disabled in the current Security state, the Effective value of HCR_EL2.FIEN is 0b1.

If [ERRIDR_EL1](#).NUM is zero, meaning no error records are implemented, or no error record accessible using System registers is owned by a node that implements the RAS Common Fault Injection Model Extension, then this bit might be RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FWB, bit [46]**When FEAT_S2FWB is implemented:**

Forced Write-Back. Defines the combined cacheability attributes in a 2 stage translation regime.

Note

When FEAT_MTE2 is implemented, if the stage 1 page or block descriptor specifies the Tagged attribute, the final memory type is Tagged only if the final cacheable memory type is Inner and Outer Write-back cacheable and the final allocation hints are Read-Allocate, Write-Allocate.

| FWB | Meaning |
|-----|--|
| 0b0 | <p>When this bit is 0, then:</p> <ul style="list-style-type: none"> The combination of stage 1 and stage 2 translations on memory type and cacheability attributes are as described in the Armv8.0 architecture. For more information, see 'Combining the stage 1 and stage 2 attributes, EL1&0 translation regime'. The encoding of the stage 2 memory type and cacheability attributes in bits[5:2] of the stage 2 page or block descriptors are as described in the Armv8.0 architecture. |
| 0b1 | <p>When this bit is 1, then:</p> <ul style="list-style-type: none"> Bit[5] of stage 2 page or block descriptor is RES0. When bit[4] of stage 2 page or block descriptor is 1 and when: <ul style="list-style-type: none"> Bits[3:2] of stage 2 page or block descriptor are 0b11, the resultant memory type and inner or outer cacheability attribute is the same as the stage 1 memory type and inner or outer cacheability attribute. Bits[3:2] of stage 2 page or block descriptor are 0b10, the resultant memory type and attribute is Normal Write-Back. Bits[3:2] of stage 2 page or block descriptor are 0b0x, the resultant memory type will be Normal Non-cacheable except where the stage 1 memory type was Device-<attr> the resultant memory type will be Device-<attr> When bit[4] of stage 2 page or block descriptor is 0 the memory type is Device, and when: <ul style="list-style-type: none"> Bits[3:2] of stage 2 page or block descriptor are 0b00, the stage 2 memory type is Device-nGnRnE. Bits[3:2] of stage 2 page or block descriptor are 0b01, the stage 2 memory type is Device-nGnRE. Bits[3:2] of stage 2 page or block descriptor are 0b10, the stage 2 memory type is Device-nGRE. Bits[3:2] of stage 2 page or block descriptor are 0b11, the stage 2 memory type is Device-GRE. If the stage 1 translation specifies a cacheable memory type, then the stage 1 cache allocation hint is applied to the final cache allocation hint where the final memory type is cacheable. If the stage 1 translation does not specify a cacheable memory type, then if the final memory type is cacheable, it is treated as read allocate, write allocate. <p>The stage 1 and stage 2 memory types are combined in the manner described in 'Combining the stage 1 and stage 2 attributes, EL1&0 translation regime'.</p> |

In Secure state, this bit applies to both the Secure stage 2 translation and the Non-secure stage 2 translation.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NV2, bit [45]**When FEAT_NV2 is implemented:**

Nested Virtualization. Changes the behaviors of HCR_EL2.{NV1, NV} to provide a mechanism for hardware to transform reads and writes from System registers into reads and writes from memory.

| NV2 | Meaning |
|-----|---|
| 0b0 | This bit has no effect on the behavior of HCR_EL2.{NV1, NV}. The behavior of HCR_EL2.{NV1, NV} is as defined for FEAT_NV. |
| 0b1 | Redefines behavior of HCR_EL2{NV1, NV} to enable: <ul style="list-style-type: none"> Transformation of read/writes to registers into read/writes to memory. Redirection of EL2 registers to EL1 registers. Any exception taken from EL1 and taken to EL1 causes SPSR_EL1.M[3:2] to be set to 0b10 and not 0b01. |

When HCR_EL2.NV is 0, the Effective value of this field is 0 and this field is treated as 0 for all purposes other than direct reads and writes of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AT, bit [44]**When FEAT_NV is implemented:**

Address Translation. EL1 execution of the following address translation instructions is trapped to EL2, when EL2 is enabled in the current Security state, reported using EC syndrome value 0x18:

- [AT S1E0R](#), [AT S1E0W](#), [AT S1E1R](#), [AT S1E1W](#), [AT S1E1RP](#), [AT S1E1WP](#).

| AT | Meaning |
|-----|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 execution of the specified instructions is trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NV1, bit [43]**When FEAT_NV2 is implemented:**

Nested Virtualization.

| NV1 | Meaning |
|-----|--|
| 0b0 | If HCR_EL2.{NV2, NV} are both 1, accesses executed from EL1 to implemented EL12, EL02, or EL2 registers are transformed to loads and stores. If HCR_EL2.NV2 is 0 or HCR_EL2.{NV2, NV} == {1, 0}, this control does not cause any instructions to be trapped. |
| 0b1 | If HCR_EL2.NV2 is 1, accesses executed from EL1 to implemented EL2 registers are transformed to loads and stores. If HCR_EL2.NV2 is 0, EL1 accesses to VBAR_EL1 , ELR_EL1 , SPSR_EL1 , and, when FEAT_CSV2_2 or FEAT_CSV2_1p2 is implemented, SCXTNUM_EL1 , are trapped to EL2, when EL2 is enabled in the current Security state, and are reported using EC syndrome value 0x18. |

If HCR_EL2.NV2 is 1, the value of HCR_EL2.NV1 defines which EL1 register accesses are transformed to loads and stores. These transformed accesses have priority over the trapping of registers.

The trapping of EL1 registers caused by other control bits has priority over the transformation of these accesses.

If a register is specified that is not implemented by an implementation, then access to that register are UNDEFINED.

For the list of registers affected, see 'Enhanced support for nested virtualization'.

If HCR_EL2.{NV1, NV} is {0, 1}, any exception taken from EL1, and taken to EL1, causes the [SPSR_EL1](#).M[3:2] to be set to 0b10, and not 0b01.

If HCR_EL2.{NV1, NV} is {1, 1}, then:

- The EL1 translation table Block and Page descriptors:
 - Bit[54] holds the PXN instead of the UXN.
 - Bit[53] is RES0.
 - Bit[6] is treated as 0 regardless of the actual value.
- If Hierarchical Permissions are enabled, the EL1 translation table Table descriptors are as follows:
 - Bit[61] is treated as 0 regardless of the actual value.
 - Bit[60] holds the PXNTable instead of the UXNTable.
 - Bit[59] is RES0.
- When executing at EL1, the PSTATE.PAN bit is treated as zero for all purposes except reading the value of the bit.
- When executing at EL1, the LDTR* instructions are treated as the equivalent LDR* instructions, and the STTR* instructions are treated as the equivalent STR* instructions.

If HCR_EL2.{NV1, NV} are {1, 0}, then the behavior is a CONSTRAINED UNPREDICTABLE choice of:

- Behaving as if HCR_EL2.NV is 1 and HCR_EL2.NV1 is 1 for all purposes other than reading back the value of the HCR_EL2.NV bit.
- Behaving as if HCR_EL2.NV is 0 and HCR_EL2.NV1 is 0 for all purposes other than reading back the value of the HCR_EL2.NV1 bit.
- Behaving with regard to the HCR_EL2.NV and HCR_EL2.NV1 bits behavior as defined in the rest of this description.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_NV is implemented:

Nested Virtualization. EL1 accesses to certain registers are trapped to EL2, when EL2 is enabled in the current Security state.

| NV1 | Meaning |
|-----|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 accesses to VBAR_EL1 , ELR_EL1 , SPSR_EL1 , and, when FEAT_CSV2_2 or FEAT_CSV2_1p2 is implemented, SCXTNUM_EL1 , are trapped to EL2, when EL2 is enabled in the current Security state, and are reported using EC syndrome value 0x18. |

If HCR_EL2.NV is 1 and HCR_EL2.NV1 is 0, then the following effects also apply:

- Any exception taken from EL1, and taken to EL1, causes the [SPSR_EL1.M\[3:2\]](#) to be set to 0b10, and not 0b01.

If HCR_EL2.NV and HCR_EL2.NV1 are both set to 1, then the following effects also apply:

- The EL1 translation table Block and Page descriptors:
 - Bit[54] holds the PXN instead of the UXN.
 - Bit[53] is RES0.
 - Bit[6] is treated as 0 regardless of the actual value.
- If Hierarchical Permissions are enabled, the EL1 translation table Table descriptors are as follows:
 - Bit[61] is treated as 0 regardless of the actual value.
 - Bit[60] holds the PXNTable instead of the UXNTable.
 - Bit[59] is RES0.
- When executing at EL1, the PSTATE.PAN bit is treated as zero for all purposes except reading the value of the bit.
- When executing at EL1, the LDTR* instructions are treated as the equivalent LDR* instructions, and the STTR* instructions are treated as the equivalent STR* instructions.

If HCR_EL2.NV is 0 and HCR_EL2.NV1 is 1, then the behavior is a CONSTRAINED UNPREDICTABLE choice of:

- Behaving as if HCR_EL2.NV is 1 and HCR_EL2.NV1 is 1 for all purposes other than reading back the value of the HCR_EL2.NV bit.
- Behaving as if HCR_EL2.NV is 0 and HCR_EL2.NV1 is 0 for all purposes other than reading back the value of the HCR_EL2.NV1 bit.
- Behaving with regard to the HCR_EL2.NV and HCR_EL2.NV1 bits behavior as defined in the rest of this description.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NV, bit [42]

When FEAT_NV2 is implemented:

Nested Virtualization.

When HCR_EL2.NV2 is 1, redefines register accesses so that:

- Instructions accessing the Special purpose registers [SPSR_EL2](#) and [ELR_EL2](#) instead access [SPSR_EL1](#) and [ELR_EL1](#) respectively.
- Instructions accessing the System registers [ESR_EL2](#) and [FAR_EL2](#) instead access [ESR_EL1](#) and [FAR_EL1](#).

When HCR_EL2.NV2 is 0, or if FEAT_NV2 is not implemented, traps functionality that is permitted at EL2 and would be UNDEFINED at EL1 if this field was 0, when EL2 is enabled in the current Security state. This applies to the following operations:

- EL1 accesses to Special-purpose registers that are not UNDEFINED at EL2.
- EL1 accesses to System registers that are not UNDEFINED at EL2.
- Execution of EL1 or EL2 translation regime address translation and TLB maintenance instructions for EL2 and above.

| NV | Meaning |
|-----|---|
| 0b0 | When this bit is set to 0, then the PE behaves as if HCR_EL2.NV2 is 0 for all purposes other than reading this register. This control does not cause any instructions to be trapped. When HCR_EL2.NV2 is 1, no FEAT_NV2 functionality is implemented. |
| 0b1 | When HCR_EL2.NV2 is 0, or if FEAT_NV2 is not implemented, EL1 accesses to the specified registers or the execution of the specified instructions are trapped to EL2, when EL2 is enabled in the current Security state. EL1 read accesses to the CurrentEL register return a value of 0x2. When HCR_EL2.NV2 is 1, this control redefines EL1 register accesses so that instructions accessing SPSR_EL2 , ELR_EL2 , ESR_EL2 , and FAR_EL2 instead access SPSR_EL1 , ELR_EL1 , ESR_EL1 , and FAR_EL1 respectively. |

When HCR_EL2.NV2 is 0, or if FEAT_NV2 is not implemented, then:

- The System or Special-purpose registers for which accesses are trapped and reported using EC syndrome value 0x18 are as follows:
 - Registers accessed using MRS or MSR with a name ending in _EL2, except [SP_EL2](#).
 - Registers accessed using MRS or MSR with a name ending in _EL12.
 - Registers accessed using MRS or MSR with a name ending in _EL02.
 - Special-purpose registers [SPSR_irq](#), [SPSR_abt](#), [SPSR_und](#) and [SPSR_fiq](#), accessed using MRS or MSR.
 - Special-purpose register [SP_EL1](#) accessed using the dedicated MRS or MSR instruction.
- The instructions for which the execution is trapped and reported using EC syndrome value 0x18 are as follows:
 - EL2 translation regime Address Translation instructions and TLB maintenance instructions.
 - EL1 translation regime Address Translation instructions and TLB maintenance instructions that are accessible only from EL2 and EL3.
- The instructions for which the execution is trapped as follows:
 - SMC in an implementation that does not include EL3 and when HCR_EL2.TSC is 1. HCR_EL2.TSC bit is not RES0 in this case. This is reported using EC syndrome value 0x17.
 - The ERET, ERETAA, and ERETAB instructions, reported using EC syndrome value 0x1A.

Note

The priority of this trap is higher than the priority of the HCR_EL2.API trap. If both of these bits are set so that EL1 execution of an ERETAA or ERETAB instruction is trapped to EL2, then the syndrome reported is 0x1A.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_NV is implemented:

Nested Virtualization. Traps functionality that is permitted at EL2 and would be UNDEFINED at EL1 if this field was 0, when EL2 is enabled in the current Security state. This applies to the following operations:

- EL1 accesses to Special-purpose registers that are not UNDEFINED at EL2.
- EL1 accesses to System registers that are not UNDEFINED at EL2.
- Execution of EL1 or EL2 translation regime address translation and TLB maintenance instructions for EL2 and above.

| NV | Meaning |
|-----|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 accesses to the specified registers or the execution of the specified instructions are trapped to EL2, when EL2 is enabled in the current Security state. EL1 read accesses to the CurrentEL register return a value of 0x2. |

The System or Special-purpose registers for which accesses are trapped and reported using EC syndrome value 0x18 are as follows:

- Registers accessed using MRS or MSR with a name ending in _EL2, except [SP_EL2](#).

- Registers accessed using MRS or MSR with a name ending in _EL12.
- Registers accessed using MRS or MSR with a name ending in _EL02.
- Special-purpose registers [SPSR_irq](#), [SPSR_abt](#), [SPSR_und](#) and [SPSR_fiq](#), accessed using MRS or MSR.
- Special-purpose register [SP_EL1](#) accessed using the dedicated MRS or MSR instruction.

The instructions for which the execution is trapped and reported using EC syndrome value 0x18 are as follows:

- EL2 translation regime Address Translation instructions and TLB maintenance instructions.
- EL1 translation regime Address Translation instructions and TLB maintenance instructions that are accessible only from EL2 and EL3.

The execution of the ERET, ERETAA, and ERETAB instructions are trapped and reported using EC syndrome value 0x1A.

Note

The priority of this trap is higher than the priority of the HCR_EL2.API trap. If both of these bits are set so that EL1 execution of an ERETAA or ERETAB instruction is trapped to EL2, then the syndrome reported is 0x1A.

The execution of the SMC instructions in an implementation that does not include EL3 and when HCR_EL2.TSC is 1 are trapped and reported using EC syndrome value 0x17. HCR_EL2.TSC bit is not RES0 in this case.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

API, bit [41]

When FEAT_PAuth is implemented:

Controls the use of instructions related to Pointer Authentication:

- In EL0, when HCR_EL2.TGE==0 or HCR_EL2.E2H==0, and the associated [SCTLR_EL1](#).En<N><M>==1.
- In EL1, the associated [SCTLR_EL1](#).En<N><M>==1.

Traps are reported using EC syndrome value 0x09. The Pointer Authentication instructions trapped are:

- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB.
- PACGA, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZB.
- RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ.
- ERETAA, ERETAB, LDRAA, and LDRAB.

| API | Meaning |
|-----|---|
| 0b0 | <p>The instructions related to Pointer Authentication are trapped to EL2, when EL2 is enabled in the current Security state and the instructions are enabled for the EL1&0 translation regime, from:</p> <ul style="list-style-type: none"> • EL0 when HCR_EL2.TGE==0 or HCR_EL2.E2H==0. • EL1. <p>If HCR_EL2.NV is 1, the HCR_EL2.NV trap takes precedence over the HCR_EL2.API trap for the ERETAA and ERETAB instructions.</p> <p>If EL2 is implemented and enabled in the current Security state and HFGITR_EL2.ERET == 1, execution at EL1 using AArch64 of ERETAA or ERETAB instructions is reported with EC syndrome value 0x1A with its associated ISS field, as the fine-grained trap has higher priority than the HCR_EL2.API == 0.</p> |
| 0b1 | This control does not cause any instructions to be trapped. |

If FEAT_PAuth is implemented but EL2 is not implemented or disabled in the current Security state, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APK, bit [40]

When FEAT_PAuth is implemented:

Trap registers holding "key" values for Pointer Authentication. Traps accesses to the following registers from EL1 to EL2, when EL2 is enabled in the current Security state, reported using EC syndrome value 0x18:

- [APIAKeyLo_EL1](#), [APIAKeyHi_EL1](#), [APIBKeyLo_EL1](#), [APIBKeyHi_EL1](#), [APDAKeyLo_EL1](#), [APDAKeyHi_EL1](#), [APDBKeyLo_EL1](#), [APDBKeyHi_EL1](#), [APGAKeyLo_EL1](#), and [APGAKeyHi_EL1](#).

| APK | Meaning |
|-----|---|
| 0b0 | Access to the registers holding "key" values for pointer authentication from EL1 are trapped to EL2, when EL2 is enabled in the current Security state. |
| 0b1 | This control does not cause any instructions to be trapped. |

Note

If FEAT_PAuth is implemented but EL2 is not implemented or is disabled in the current Security state, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME, bit [39]

When FEAT_TME is implemented:

Enables access to the TSTART, TCOMMIT, TTEST, and TCANCEL instructions at EL0 and EL1.

| TME | Meaning |
|-----|---|
| 0b0 | EL0 and EL1 accesses to TSTART, TCOMMIT, TTEST, and TCANCEL instructions are UNDEFINED. |
| 0b1 | This control does not cause any instruction to be UNDEFINED. |

If EL2 is not implemented or is disabled in the current Security state, the Effective value of this bit is 0b1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MIOCNCNCE, bit [38]

Mismatched Inner/Outer Cacheable Non-Coherency Enable, for the EL1&0 translation regimes.

| MIOCNCNCE | Meaning |
|------------------|--|
| 0b0 | For the EL1&0 translation regimes, for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there must be no loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute. |
| 0b1 | For the EL1&0 translation regimes, for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there might be a loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute. |

For more information, see 'Mismatched memory attributes'.

This field can be implemented as RAZ/WI.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TEA, bit [37]

When FEAT_RAS is implemented:

Route synchronous External abort exceptions to EL2.

| TEA | Meaning |
|------------|---|
| 0b0 | This control does not cause exceptions to be routed from EL0 and EL1 to EL2. |
| 0b1 | Route synchronous External abort exceptions from EL0 and EL1 to EL2, when EL2 is enabled in the current Security state, if not routed to EL3. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TERR, bit [36]

When FEAT_RAS is implemented:

Trap Error record accesses. Trap accesses to the RAS error registers from EL1 to EL2 as follows:

- If EL1 is using AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x18:
 - [ERRIDR_EL1](#), [ERRSELR_EL1](#), [ERXADDR_EL1](#), [ERXCTLR_EL1](#), [ERXFR_EL1](#), [ERXMISC0_EL1](#), [ERXMISC1_EL1](#), and [ERXSTATUS_EL1](#).
 - When FEAT_RASv1p1 is implemented, [ERXMISC2_EL1](#), and [ERXMISC3_EL1](#).
- If EL1 is using AArch32 state, MCR or MRC accesses are trapped to EL2, reported using EC syndrome value 0x03, MCRR or MRRC accesses are trapped to EL2, reported using EC syndrome value 0x04:
 - [ERRIDR](#), [ERRSELR](#), [ERXADDR](#), [ERXADDR2](#), [ERXCTLR](#), [ERXCTLR2](#), [ERXFR](#), [ERXFR2](#), [ERXMISC0](#), [ERXMISC1](#), [ERXMISC2](#), [ERXMISC3](#), and [ERXSTATUS](#).
 - When FEAT_RASv1p1 is implemented, [ERXMISC4](#), [ERXMISC5](#), [ERXMISC6](#), and [ERXMISC7](#).

| TERR | Meaning |
|-------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Accesses to the specified registers from EL1 generate a Trap exception to EL2, when EL2 is enabled in the current Security state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLOR, bit [35]

When FEAT_LOR is implemented:

Trap LOR registers. Traps Non-secure EL1 accesses to [LORSA_EL1](#), [LOREA_EL1](#), [LORN_EL1](#), [LORC_EL1](#), and [LORID_EL1](#) registers to EL2.

| TLOR | Meaning |
|-------------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 accesses to the LOR registers are trapped to EL2. |

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E2H, bit [34]

When FEAT_VHE is implemented:

EL2 Host. Enables a configuration where a Host Operating System is running in EL2, and the Host Operating System's applications are running in EL0.

| E2H | Meaning |
|------------|--|
| 0b0 | The facilities to support a Host Operating System at EL2 are disabled. |
| 0b1 | The facilities to support a Host Operating System at EL2 are enabled. |

For information on the behavior of this bit see 'Behavior of HCR_EL2.E2H'.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ID, bit [33]

Stage 2 Instruction access cacheability disable. For the EL1&0 translation regime, when EL2 is enabled in the current Security state and HCR_EL2.VM==1, this control forces all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable.

| ID | Meaning |
|-----|--|
| 0b0 | This control has no effect on stage 2 of the EL1&0 translation regime. |
| 0b1 | Forces all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable. |

This bit has no effect on the EL2, EL2&0, or EL3 translation regimes.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CD, bit [32]

Stage 2 Data access cacheability disable. For the EL1&0 translation regime, when EL2 is enabled in the current Security state and HCR_EL2.VM==1, this control forces all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable.

| CD | Meaning |
|-----|--|
| 0b0 | This control has no effect on stage 2 of the EL1&0 translation regime for data accesses and translation table walks. |
| 0b1 | Forces all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable. |

This bit has no effect on the EL2, EL2&0, or EL3 translation regimes.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RW, bit [31]

When EL1 is capable of using AArch32:

Execution state control for lower Exception levels:

| RW | Meaning |
|-----|---|
| 0b0 | Lower levels are all AArch32. |
| 0b1 | The Execution state for EL1 is AArch64. The Execution state for EL0 is determined by the current value of PSTATE.nRW when executing at EL0. |

In an implementation that includes EL3, when EL2 is not enabled in Secure state, the PE behaves as if this bit has the same value as the [SCR_EL3.RW](#) bit for all purposes other than a direct read or write access of HCR_EL2.

The RW bit is permitted to be cached in a TLB.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 1 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAO/WI.

TRVM, bit [30]

Trap Reads of Virtual Memory controls. Traps EL1 reads of the virtual memory control registers to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL1 is using AArch64 state, the following registers are trapped to EL2 and reported using EC syndrome value 0x18.
 - [SCTLR_EL1](#), [TTBR0_EL1](#), [TTBR1_EL1](#), [TCR_EL1](#), [ESR_EL1](#), [FAR_EL1](#), [AFSR0_EL1](#), [AFSR1_EL1](#), [MAIR_EL1](#), [AMAIR_EL1](#), [CONTEXTIDR_EL1](#).
- If EL1 is using AArch32 state, accesses using MRC to the following registers are trapped to EL2 and reported using EC syndrome value 0x03, accesses using MRRC are trapped to EL2 and reported using EC syndrome value 0x04:
 - [SCTLR](#), [TTBR0](#), [TTBR1](#), [TTBCR](#), [TTBCR2](#), [DACR](#), [DFSR](#), [ISFR](#), [DFAR](#), [IFAR](#), [ADFSR](#), [AIFSR](#), [PRRR](#), [NMRR](#), [MAIR0](#), [MAIR1](#), [AMAIR0](#), [AMAIR1](#), [CONTEXTIDR](#).

| TRVM | Meaning |
|------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 read accesses to the specified Virtual Memory controls are trapped to EL2, when EL2 is enabled in the current Security state. |

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

Note

EL2 provides a second stage of address translation, that a hypervisor can use to remap the address map defined by a Guest OS. In addition, a hypervisor can trap attempts by a Guest OS to write to the registers that control the memory system. A hypervisor might use this trap as part of its virtualization of memory management.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HCD, bit [29]

When EL3 is not implemented:

HVC instruction disable. Disables EL1 execution of HVC instructions, from both Execution states, when EL2 is enabled in the current Security state, reported using EC syndrome value 0x00.

| HCD | Meaning |
|-----|--|
| 0b0 | HVC instruction execution is enabled at EL2 and EL1. |
| 0b1 | HVC instructions are UNDEFINED at EL2 and EL1. Any resulting exception is taken to the Exception level at which the HVC instruction is executed. |

Note

HVC instructions are always UNDEFINED at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TDZ, bit [28]

Trap [DC ZVA](#) instructions. Traps EL0 and EL1 execution of [DC ZVA](#) instructions to EL2, when EL2 is enabled in the current Security state, from AArch64 state only, reported using EC syndrome value 0x18.

If FEAT_MTE is implemented, this trap also applies to [DC GVA](#) and [DC GZVA](#).

| TDZ | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | In AArch64 state, any attempt to execute an instruction this trap applies to at EL1, or at EL0 when the instruction is not UNDEFINED at EL0, is trapped to EL2 when EL2 is enabled in the current Security state. Reading the DCZID_EL0 returns a value that indicates that the instructions this trap applies to are not supported. |

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TGE, bit [27]

Trap General Exceptions, from EL0.

| TGE | Meaning |
|-----|---|
| 0b0 | This control has no effect on execution at EL0. |
| 0b1 | When EL2 is not enabled in the current Security state, this control has no effect on execution at EL0. When EL2 is enabled in the current Security state, in all cases: <ul style="list-style-type: none"> All exceptions that would be routed to EL1 are routed to EL2. If EL1 is using AArch64, the SCTLR_EL1.M field is treated as being 0 for all purposes other than returning the result of a direct read of SCTLR_EL1. If EL1 is using AArch32, the SCTLR.M field is treated as being 0 for all purposes other than returning the result of a direct read of SCTLR. All virtual interrupts are disabled. Any IMPLEMENTATION DEFINED mechanisms for signaling virtual interrupts are disabled. An exception return to EL1 is treated as an illegal exception return. The MDCR_EL2.{TDRA, TDOSA, TDA, TDE} fields are treated as being 1 for all purposes other than returning the result of a direct read of MDCR_EL2. In addition, when EL2 is enabled in the current Security state, if: <ul style="list-style-type: none"> HCR_EL2.E2H is 0, the Effective values of the HCR_EL2.{FMO, IMO, AMO} fields are 1. HCR_EL2.E2H is 1, the Effective values of the HCR_EL2.{FMO, IMO, AMO} fields are 0. For further information on the behavior of this bit when E2H is 1, see 'Behavior of HCR_EL2.E2H'. |

HCR_EL2.TGE must not be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TVM, bit [26]

Trap Virtual Memory controls. Traps EL1 writes to the virtual memory control registers to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL1 is using AArch64 state, the following registers are trapped to EL2 and reported using EC syndrome value 0x18:
 - [SCTLR_EL1](#), [TTBR0_EL1](#), [TTBR1_EL1](#), [TCR_EL1](#), [ESR_EL1](#), [FAR_EL1](#), [AFSR0_EL1](#), [AFSR1_EL1](#), [MAIR_EL1](#), [AMAIR_EL1](#), [CONTEXTIDR_EL1](#).
- If EL1 is using AArch32 state, accesses using MCR to the following registers are trapped to EL2 and reported using EC syndrome value 0x03, accesses using MCRR are trapped to EL2 and reported using EC syndrome value 0x04:
 - [SCTLR](#), [TTBR0](#), [TTBR1](#), [TTBCR](#), [TTBCR2](#), [DACR](#), [DFSR](#), [IFSR](#), [DFAR](#), [IFAR](#), [ADFSR](#), [AIFSR](#), [PRRR](#), [NMRR](#), [MAIR0](#), [MAIR1](#), [AMAIR0](#), [AMAIR1](#), [CONTEXTIDR](#).

| TVM | Meaning |
|------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 write accesses to the specified EL1 virtual memory control registers are trapped to EL2, when EL2 is enabled in the current Security state. |

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TTLB, bit [25]

Trap TLB maintenance instructions. Traps EL1 execution of TLB maintenance instructions to EL2, when EL2 is enabled in the current Security state, as follows:

- When EL1 is using AArch64 state, the following instructions are trapped to EL2 and reported using EC syndrome value 0x18:
 - [TLBI VMALLE1](#), [TLBI VAE1](#), [TLBI ASIDE1](#), [TLBI VAAE1](#), [TLBI VALE1](#), [TLBI VAALE1](#).
 - [TLBI VMALLE1IS](#), [TLBI VAE1IS](#), [TLBI ASIDE1IS](#), [TLBI VAAE1IS](#), [TLBI VALE1IS](#), [TLBI VAALE1IS](#).
 - If FEAT_TLBIOS is implemented, this trap applies to [TLBI VMALLE1OS](#), [TLBI VAE1OS](#), [TLBI ASIDE1OS](#), [TLBI VAAE1OS](#), [TLBI VALE1OS](#), [TLBI VAALE1OS](#).
 - If FEAT_TLBIRANGE is implemented, this trap applies to [TLBI RVAE1](#), [TLBI RVAAE1](#), [TLBI RVALE1](#), [TLBI RVALE1IS](#), [TLBI RVAAE1IS](#), [TLBI RVALE1IS](#), [TLBI RVAAE1IS](#).
 - If FEAT_TLBIOS and FEAT_TLBIRANGE are implemented, this trap applies to [TLBI RVAE1OS](#), [TLBI RVAAE1OS](#), [TLBI RVALE1OS](#), [TLBI RVALE1OS](#).
- When EL1 is using AArch32 state, the following instructions are trapped to EL2 and reported using EC syndrome value 0x03:
 - [TLBIALLIS](#), [TLBIMVAIS](#), [TLBIASIDIS](#), [TLBIMVAAIS](#), [TLBIMVALIS](#), [TLBIMVAALIS](#).
 - [TLBIALL](#), [TLBIMVA](#), [TLBIASID](#), [TLBIMVAA](#), [TLBIMVAL](#), [TLBIMVAAL](#).
 - [ITLBIALL](#), [ITLBIMVA](#), [ITLBIASID](#).
 - [DTLBIALL](#), [DTLBIMVA](#), [DTLBIASID](#).

| TTLB | Meaning |
|-------------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 execution of the specified TLB maintenance instructions are trapped to EL2, when EL2 is enabled in the current Security state. |

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

Note

The TLB maintenance instructions are UNDEFINED at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TPU, bit [24]

Trap cache maintenance instructions that operate to the Point of Unification. Traps execution of those cache maintenance instructions to EL2, when EL2 is enabled in the current Security state as follows:

- If EL0 is using AArch64 state and the value of [SCTLR_EL1](#).UCI is not 0, the following instructions are trapped to EL2 and reported with EC syndrome value 0x18:
 - [IC IVAU](#), [DC CVAU](#). If the value of [SCTLR_EL1](#).UCI is 0 these instructions are UNDEFINED at EL0 and any resulting exception is higher priority than this trap to EL2.
- If EL1 is using AArch64 state, the following instructions are trapped to EL2 and reported with EC syndrome value 0x18:
 - [IC IVAU](#), [IC IALLU](#), [IC IALLUIS](#), [DC CVAU](#).
- If EL1 is using AArch32 state, the following instructions are trapped to EL2 and reported with EC syndrome value 0x18:
 - [ICIMVAU](#), [IC IALLU](#), [IC IALLUIS](#), [DCCMVAU](#).

Note

An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2. In addition:

- [IC IALLUIS](#) and [IC IALLU](#) are always UNDEFINED at EL0 using AArch64.
- [ICIMVAU](#), [IC IALLU](#), [IC IALLUIS](#), and [DCCMVAU](#) are always UNDEFINED at EL0 using AArch32.

| TPU | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Execution of the specified instructions is trapped to EL2, when EL2 is enabled in the current Security state. |

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TPCP, bit [23]

When FEAT_DPB is implemented:

Trap data or unified cache maintenance instructions that operate to the Point of Coherency or Persistence. Traps execution of those cache maintenance instructions to EL2, when EL2 is enabled in the current Security state as follows:

- If EL0 is using AArch64 state and the value of [SCTLR_EL1](#).UCI is not 0, the following instructions are trapped to EL2 and reported using EC syndrome value 0x18:
 - [DC CIVAC](#), [DC CVAC](#), [DC CVAP](#). If the value of [SCTLR_EL1](#).UCI is 0 these instructions are UNDEFINED at EL0 and any resulting exception is higher priority than this trap to EL2.
- If EL1 is using AArch64 state, the following instructions are trapped to EL2 and reported using EC syndrome value 0x18:
 - [DC IVAC](#), [DC CIVAC](#), [DC CVAC](#), [DC CVAP](#).
- If EL1 is using AArch32 state, the following instructions are trapped to EL2 and reported using EC syndrome value 0x03:
 - [DCIMVAC](#), [DCCIMVAC](#), [DCCMVAC](#).

If FEAT_DPB2 is implemented, this trap also applies to [DC CVADP](#).

If FEAT_MTE is implemented, this trap also applies to [DC CIGVAC](#), [DC CIGDVAC](#), [DC IGVAC](#), [DC IGDVAC](#), [DC CGVAC](#), [DC CGDVAC](#), [DC CGVAP](#) and [DC CGDVAP](#).

If FEAT_DPB2 and FEAT_MTE are implemented, this trap also applies to [DC CGVADP](#) and [DC CGDVADP](#).

Note

- An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2. In addition:
 - AArch64 instructions which invalidate by VA to the Point of Coherency are always UNDEFINED at EL0 using AArch64.
 - [DCIMVAC](#), [DCCIMVAC](#), and [DCCMVAC](#) are always UNDEFINED at EL0 using AArch32.
- In Armv8.0 and Armv8.1, this field is named TPC. From Armv8.2, it is named TPCP.

| TPCP | Meaning |
|------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Execution of the specified instructions is trapped to EL2, when EL2 is enabled in the current Security state. |

If the Point of Coherency is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean, invalidate, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

If HCR_EL2.{E2H, TGE} is set to {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Trap data or unified cache maintenance instructions that operate to the Point of Coherency. Traps execution of those cache maintenance instructions to EL2, when EL2 is enabled in the current Security state as follows:

- If EL0 is using AArch64 state and the value of [SCTLR_EL1](#).UCI is not 0, accesses to the following registers are trapped and reported using EC syndrome value 0x18:
 - [DC CIVAC](#), [DC CVAC](#). However, if the value of [SCTLR_EL1](#).UCI is 0 these instructions are UNDEFINED at EL0 and any resulting exception is higher priority than this trap to EL2.
- If EL1 is using AArch64 state, accesses to [DC IVAC](#), [DC CIVAC](#), [DC CVAC](#) are trapped and reported using EC syndrome value 0x18.
- When EL1 is using AArch32, accesses to [DCIMVAC](#), [DCCIMVAC](#), and [DCCMVAC](#) are trapped and reported using EC syndrome value 0x03.

Note

- An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2. In addition:
 - AArch64 instructions which invalidate by VA to the Point of Coherency are always UNDEFINED at EL0 using AArch64.
 - [DCIMVAC](#), [DCCIMVAC](#), and [DCCMVAC](#) are always UNDEFINED at EL0 using AArch32.
- In Armv8.0 and Armv8.1, this field is named TPC. From Armv8.2, it is named TPCP.

| TPC | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Execution of the specified instructions is trapped to EL2, when EL2 is enabled in the current Security state. |

If the Point of Coherency is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean, invalidate, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TSW, bit [22]

Trap data or unified cache maintenance instructions that operate by Set/Way. Traps execution of those cache maintenance instructions at EL1 to EL2, when EL2 is enabled in the current Security state as follows:

- If EL1 is using AArch64 state, accesses to [DC ISW](#), [DC CSW](#), [DC CISW](#) are trapped to EL2, reported using EC syndrome value 0x18.
- If EL1 is using AArch32 state, accesses to [DCISW](#), [DCCSW](#), [DCCISW](#) are trapped to EL2, reported using EC syndrome value 0x03.

If FEAT_MTE2 is implemented, this trap also applies to [DC IGSW](#), [DC IGDSW](#), [DC CGSW](#), [DC CGDW](#), [DC CIGSW](#), and [DC CIGDSW](#).

Note

An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2, and these instructions are always UNDEFINED at EL0.

| TSW | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Execution of the specified instructions is trapped to EL2, when EL2 is enabled in the current Security state. |

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TACR, bit [21]

Trap Auxiliary Control Registers. Traps EL1 accesses to the Auxiliary Control Registers to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL1 is using AArch64 state, accesses to [ACTLR_EL1](#) to EL2, are trapped to EL2 and reported using EC syndrome value 0x18.
- If EL1 is using AArch32 state, accesses to [ACTLR](#) and, if implemented, [ACTLR2](#) are trapped to EL2 and reported using EC syndrome value 0x03.

| TACR | Meaning |
|------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 accesses to the specified registers are trapped to EL2, when EL2 is enabled in the current Security state. |

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

Note

[ACTLR_EL1](#) is not accessible at EL0.

[ACTLR](#) and [ACTLR2](#) are not accessible at EL0.

The Auxiliary Control Registers are IMPLEMENTATION DEFINED registers that might implement global control bits for the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TIDCP, bit [20]

Trap IMPLEMENTATION DEFINED functionality. Traps EL1 accesses to the encodings reserved for IMPLEMENTATION DEFINED functionality to EL2, when EL2 is enabled in the current Security state as follows:

- In AArch64 state, access to any of the encodings in the following reserved encoding spaces are trapped and reported using EC syndrome 0x18:
 - IMPLEMENTATION DEFINED System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}.
 - IMPLEMENTATION DEFINED System registers, which are accessed using MRS and MSR with the [S3_<op1>_<Cn>_<Cm>_<op2>](#) register name.
- In AArch32 state, MCR and MRC access to instructions with the following encodings are trapped and reported using EC syndrome 0x03:
 - All coproc==p15, CRn==c9, opc1 == {0-7}, CRm == {c0-c2, c5-c8}, opc2 == {0-7}.
 - All coproc==p15, CRn==c10, opc1 == {0-7}, CRm == {c0, c1, c4, c8}, opc2 == {0-7}.
 - All coproc==p15, CRn==c11, opc1=={0-7}, CRm == {c0-c8, c15}, opc2 == {0-7}.

When the value of HCR_EL2.TIDCP is 1, it is IMPLEMENTATION DEFINED whether any of this functionality accessed from EL0 is trapped to EL2. If it is not, then it is UNDEFINED, and any attempt to access it from EL0 generates an exception that is taken to EL1.

| TIDCP | Meaning |
|-------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 accesses to or execution of the specified encodings reserved for IMPLEMENTATION DEFINED functionality are trapped to EL2, when EL2 is enabled in the current Security state. |

An implementation can also include IMPLEMENTATION DEFINED registers that provide additional controls, to give finer-grained control of the trapping of IMPLEMENTATION DEFINED features.

Note

Arm expects the trapping of EL0 accesses to these functions to EL2 to be unusual, and used only when the hypervisor is virtualizing EL0 operation. Arm strongly recommends that unless the hypervisor must virtualize EL0 operation, an EL0 access to any of these functions is UNDEFINED, as it would be if the implementation did not include EL2. The PE then takes any resulting exception to EL1.

The trapping of accesses to these registers from EL1 is higher priority than an exception resulting from the register access being UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TSC, bit [19]

Trap SMC instructions. Traps EL1 execution of SMC instructions to EL2, when EL2 is enabled in the current Security state.

If execution is in AArch64 state, the trap is reported using EC syndrome value 0x17.

If execution is in AArch32 state, the trap is reported using EC syndrome value 0x13.

Note

HCR_EL2.TSC traps execution of the SMC instruction. It is not a routing control for the SMC exception. Trap exceptions and SMC exceptions have different preferred return addresses.

| TSC | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | <p>If EL3 is implemented, then any attempt to execute an SMC instruction at EL1 is trapped to EL2, when EL2 is enabled in the current Security state, regardless of the value of SCR_EL3.SMD.</p> <p>If EL3 is not implemented, FEAT_NV is implemented, and HCR_EL2.NV is 1, then any attempt to execute an SMC instruction at EL1 using AArch64 is trapped to EL2, when EL2 is enabled in the current Security state.</p> <p>If EL3 is not implemented, and either FEAT_NV is not implemented or HCR_EL2.NV is 0, then it is IMPLEMENTATION DEFINED whether:</p> <ul style="list-style-type: none"> Any attempt to execute an SMC instruction at EL1 is trapped to EL2, when EL2 is enabled in the current Security state. Any attempt to execute an SMC instruction is UNDEFINED. |

In AArch32 state, the Armv8-A architecture permits, but does not require, this trap to apply to conditional SMC instructions that fail their condition code check, in the same way as with traps on other conditional instructions.

SMC instructions are UNDEFINED at EL0.

If EL3 is not implemented, and either FEAT_NV is not implemented or HCR_EL2.NV is 0, then it is IMPLEMENTATION DEFINED whether this bit is:

- RES0.
- Implemented with the functionality as described in HCR_EL2.TSC.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TID3, bit [18]

Trap ID group 3. Traps EL1 reads of group 3 ID registers to EL2, when EL2 is enabled in the current Security state, as follows:

In AArch64 state:

- Reads of the following registers are trapped to EL2, reported using EC syndrome value 0x18:
 - [ID_PFR0_EL1](#), [ID_PFR1_EL1](#), [ID_PFR2_EL1](#), [ID_DFR0_EL1](#), [ID_AFR0_EL1](#), [ID_MMFR0_EL1](#), [ID_MMFR1_EL1](#), [ID_MMFR2_EL1](#), [ID_MMFR3_EL1](#), [ID_ISAR0_EL1](#), [ID_ISAR1_EL1](#), [ID_ISAR2_EL1](#), [ID_ISAR3_EL1](#), [ID_ISAR4_EL1](#), [ID_ISAR5_EL1](#), [MVFR0_EL1](#), [MVFR1_EL1](#), [MVFR2_EL1](#).
 - [ID_AA64PFR0_EL1](#), [ID_AA64PFR1_EL1](#), [ID_AA64DFR0_EL1](#), [ID_AA64DFR1_EL1](#), [ID_AA64ISAR0_EL1](#), [ID_AA64ISAR1_EL1](#), [ID_AA64MMFR0_EL1](#), [ID_AA64MMFR1_EL1](#), [ID_AA64AFR0_EL1](#), [ID_AA64AFR1_EL1](#).
 - If FEAT_FGT is implemented:
 - [ID_MMFR4_EL1](#) and [ID_MMFR5_EL1](#) are trapped to EL2.
 - [ID_AA64MMFR2_EL1](#) and [ID_ISAR6_EL1](#) are trapped to EL2.
 - [ID_DFR1_EL1](#) is trapped to EL2.
 - [ID_AA64ZFR0_EL1](#) is trapped to EL2.
 - [ID_AA64SMFR0_EL1](#) is trapped to EL2.
 - [ID_AA64ISAR2_EL1](#) is trapped to EL2.

- This field traps all MRS accesses to registers in the following range that are not already mentioned in this field description: Op0 == 3, op1 == 0, CRn == c0, CRm == {c1-c7}, op2 == {0-7}.
- If FEAT_FGT is not implemented:
 - [ID_MMFR4_EL1](#) and [ID_MMFR5_EL1](#) are trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to [ID_MMFR4_EL1](#) or [ID_MMFR5_EL1](#) are trapped to EL2.
 - [ID_AA64MMFR2_EL1](#) and [ID_ISAR6_EL1](#) are trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to [ID_AA64MMFR2_EL1](#) or [ID_ISAR6_EL1](#) are trapped to EL2.
 - [ID_DFR1_EL1](#) is trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to [ID_DFR1_EL1](#) are trapped to EL2.
 - [ID_AA64ZFR0_EL1](#) is trapped to EL2, unless implemented as RAZ then it is IMPLEMENTATION DEFINED whether accesses to [ID_AA64ZFR0_EL1](#) are trapped to EL2.
 - [ID_AA64SMFR0_EL1](#) is trapped to EL2, unless implemented as RAZ then it is IMPLEMENTATION DEFINED whether accesses to [ID_AA64SMFR0_EL1](#) are trapped to EL2.
 - [ID_AA64ISAR2_EL1](#) is trapped to EL2, unless implemented as RAZ then it is IMPLEMENTATION DEFINED whether accesses to [ID_AA64ISAR2_EL1](#) are trapped to EL2.
 - Otherwise, it is IMPLEMENTATION DEFINED whether this bit traps MRS accesses to registers in the following range that are not already mentioned in this field description: Op0 == 3, op1 == 0, CRn == c0, CRm == {c1-c7}, op2 == {0-7}.

In AArch32 state:

- VMRS access to [MVFR0](#), [MVFR1](#), and [MVFR2](#), are trapped to EL2, reported using EC syndrome value 0x08, unless access is also trapped by [HCPTR](#) which takes priority.
- MRC access to the following registers are trapped to EL2, reported using EC syndrome value 0x03:
 - [ID_PFR0](#), [ID_PFR1](#), [ID_PFR2](#), [ID_DFR0](#), [ID_AFR0](#), [ID_MMFR0](#), [ID_MMFR1](#), [ID_MMFR2](#), [ID_MMFR3](#), [ID_ISAR0](#), [ID_ISAR1](#), [ID_ISAR2](#), [ID_ISAR3](#), [ID_ISAR4](#), [ID_ISAR5](#).
 - If FEAT_FGT is implemented:
 - [ID_MMFR4](#) and [ID_MMFR5](#) are trapped to EL2.
 - [ID_ISAR6](#) is trapped to EL2.
 - [ID_DFR1](#) is trapped to EL2.
 - This field traps all MRC accesses to encodings in the following range that are not already mentioned in this field description: coproc == p15, opc1 == 0, CRn == c0, CRm == {c2-c7}, opc2 == {0-7}.
 - If FEAT_FGT is not implemented:
 - [ID_MMFR4](#) and [ID_MMFR5](#) are trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to [ID_MMFR4](#) or [ID_MMFR5](#) are trapped.
 - [ID_ISAR6](#) is trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to [ID_ISAR6](#) are trapped to EL2.
 - [ID_DFR1](#) is trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to [ID_DFR1](#) are trapped to EL2.
 - Otherwise, it is IMPLEMENTATION DEFINED whether this bit traps all MRC accesses to registers in the following range not already mentioned in this field description with coproc == p15, opc1 == 0, CRn == c0, CRm == {c2-c7}, opc2 == {0-7}.

| TID3 | Meaning |
|-------------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | The specified EL1 read accesses to ID group 3 registers are trapped to EL2, when EL2 is enabled in the current Security state. |

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TID2, bit [17]

Trap ID group 2. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state, as follows:

- If EL1 is using AArch64, reads of [CTR_EL0](#), [CCSIDR_EL1](#), [CCSIDR2_EL1](#), [CLIDR_EL1](#), and [CSSELR_EL1](#) are trapped to EL2, reported using EC syndrome value 0x18.
- If EL0 is using AArch64 and the value of [SCTLR_EL1](#).UCT is not 0, reads of [CTR_EL0](#) are trapped to EL2, reported using EC syndrome value 0x18. If the value of [SCTLR_EL1](#).UCT is 0, then EL0 reads of [CTR_EL0](#) are trapped to EL1 and the resulting exception takes precedence over this trap.
- If EL1 is using AArch64, writes to [CSSELR_EL1](#) are trapped to EL2, reported using EC syndrome value 0x18.
- If EL1 is using AArch32, reads of [CTR](#), [CCSIDR](#), [CCSIDR2](#), [CLIDR](#), and [CSSELR](#) are trapped to EL2, reported using EC syndrome value 0x03.
- If EL1 is using AArch32, writes to [CSSELR](#) are trapped to EL2, reported using EC syndrome value 0x03.

| TID2 | Meaning |
|-------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | The specified EL1 and EL0 accesses to ID group 2 registers are trapped to EL2, when EL2 is enabled in the current Security state. |

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TID1, bit [16]

Trap ID group 1. Traps EL1 reads of the following registers to EL2, when EL2 is enabled in the current Security state as follows:

- In AArch64 state, accesses of [REVIDR_EL1](#), [AIDR_EL1](#), [SMIDR_EL1](#), reported using EC syndrome value 0x18.
- In AArch32 state, accesses of [TCMTR](#), [TLBTR](#), [REVIDR](#), [AIDR](#), reported using EC syndrome value 0x03.

| TID1 | Meaning |
|-------------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | The specified EL1 read accesses to ID group 1 registers are trapped to EL2, when EL2 is enabled in the current Security state. |

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TID0, bit [15]**When AArch32 is supported:**

Trap ID group 0. Traps the following register accesses to EL2:

- EL1 reads of the [JIDR](#), reported using EC syndrome value 0x05.
- If the [JIDR](#) is RAZ from EL0, EL0 reads of the [JIDR](#), reported using EC syndrome value 0x05.
- EL1 accesses using VMRS of the [FPSID](#), reported using EC syndrome value 0x08.

Note

- It is IMPLEMENTATION DEFINED whether the [JIDR](#) is RAZ or UNDEFINED at EL0. If it is UNDEFINED at EL0, then any resulting exception takes precedence over this trap.
- The [FPSID](#) is not accessible at EL0 using AArch32.
- Writes to the [FPSID](#) are ignored, and not trapped by this control.

| TID0 | Meaning |
|------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | The specified EL1 read accesses to ID group 0 registers are trapped to EL2, when EL2 is enabled in the current Security state. |

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWE, bit [14]

Traps EL0 and EL1 execution of WFE instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states, reported using EC syndrome value 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFET instruction.

| TWE | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any attempt to execute a WFE instruction at EL0 or EL1 is trapped to EL2, when EL2 is enabled in the current Security state, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWE or SCTLR_EL1.nTWE . |

In AArch32 state, the attempted execution of a conditional WFE instruction is trapped only if the instruction passes its condition code check.

Note

Since a WFE can complete at any time, even without a Wakeup event, the traps on WFE are not guaranteed to be taken, even if the WFE is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

For more information about when WFE instructions can cause the PE to enter a low-power state, see 'Wait for Event mechanism and Send event'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TWI, bit [13]

Traps EL0 and EL1 execution of WFI instructions to EL2, when EL2 is enabled in the current Security state, from both Execution states, reported using EC syndrome value 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFIT instruction.

| TWI | Meaning |
|------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any attempt to execute a WFI instruction at EL0 or EL1 is trapped to EL2, when EL2 is enabled in the current Security state, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWI or SCTLR_EL1.nTWI . |

In AArch32 state, the attempted execution of a conditional WFI instruction is trapped only if the instruction passes its condition code check.

Note

Since a WFI can complete at any time, even without a Wakeup event, the traps on WFI are not guaranteed to be taken, even if the WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

For more information about when WFI instructions can cause the PE to enter a low-power state, see 'Wait for Interrupt'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DC, bit [12]

Default Cacheability.

| DC | Meaning |
|-----------|---|
| 0b0 | This control has no effect on the EL1&0 translation regime. |
| 0b1 | In any Security state: <ul style="list-style-type: none"> When EL1 is using AArch64, the PE behaves as if the value of the SCTLR_EL1.M field is 0 for all purposes other than returning the value of a direct read of SCTLR_EL1. When EL1 is using AArch32, the PE behaves as if the value of the SCTLR.M field is 0 for all purposes other than returning the value of a direct read of SCTLR. The PE behaves as if the value of the HCR_EL2.VM field is 1 for all purposes other than returning the value of a direct read of HCR_EL2. The memory type produced by stage 1 of the EL1&0 translation regime is Normal Non-Shareable, Inner Write-Back Read-Allocate Write-Allocate, Outer Write-Back Read-Allocate Write-Allocate. |

This field has no effect on the EL2, EL2&0, and EL3 translation regimes.

This bit is permitted to be cached in a TLB.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BSU, bits [11:10]

Barrier Shareability upgrade. This field determines the minimum shareability domain that is applied to any barrier instruction executed from EL1 or EL0:

| BSU | Meaning |
|------|------------------|
| 0b00 | No effect. |
| 0b01 | Inner Shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Full system. |

This value is combined with the specified level of the barrier held in its instruction, using the same principles as combining the shareability attributes from two stages of address translation.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0b00 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FB, bit [9]

Force broadcast. Causes the following instructions to be broadcast within the Inner Shareable domain when executed from EL1:

AArch32: [BPIALL](#), [TLBIALL](#), [TLBIMVA](#), [TLBIASID](#), [DTLBIALL](#), [DTLBIMVA](#), [DTLBIASID](#), [ITLBIALL](#), [ITLBIMVA](#), [ITLBIASID](#), [TLBIMVAA](#), [ICIALLU](#), [TLBIMVAL](#), [TLBIMVAAL](#).

AArch64: [TLBI VMALLE1](#), [TLBI VAE1](#), [TLBI ASIDE1](#), [TLBI VAAE1](#), [TLBI VALE1](#), [TLBI VAALE1](#), [IC IALLU](#), [TLBI RVAE1](#), [TLBI RVAAE1](#), [TLBI RVALE1](#), [TLBI RVAALE1](#).

| FB | Meaning |
|-----|--|
| 0b0 | This field has no effect on the operation of the specified instructions. |
| 0b1 | When one of the specified instruction is executed at EL1, the instruction is broadcast within the Inner Shareable shareability domain. |

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VSE, bit [8]

Virtual SError interrupt.

| VSE | Meaning |
|-----|--|
| 0b0 | This mechanism is not making a virtual SError interrupt pending. |
| 0b1 | A virtual SError interrupt is pending because of this mechanism. |

The virtual SError interrupt is enabled only when the value of HCR_EL2.{TGE, AMO} is {0, 1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VI, bit [7]

Virtual IRQ Interrupt.

| VI | Meaning |
|-----|---|
| 0b0 | This mechanism is not making a virtual IRQ pending. |
| 0b1 | A virtual IRQ is pending because of this mechanism. |

The virtual IRQ is enabled only when the value of HCR_EL2.{TGE, IMO} is {0, 1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VF, bit [6]

Virtual FIQ Interrupt.

| VF | Meaning |
|-----|---|
| 0b0 | This mechanism is not making a virtual FIQ pending. |
| 0b1 | A virtual FIQ is pending because of this mechanism. |

The virtual FIQ is enabled only when the value of HCR_EL2.{TGE, FMO} is {0, 1}.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AMO, bit [5]

Physical SError interrupt routing.

| AMO | Meaning |
|-----|---|
| 0b0 | When executing at Exception levels below EL2, and EL2 is enabled in the current Security state: <ul style="list-style-type: none"> • When the value of HCR_EL2.TGE is 0, Physical SError interrupts are not taken to EL2. • When the value of HCR_EL2.TGE is 1, Physical SError interrupts are taken to EL2 unless they are routed to EL3. • Virtual SError interrupts are disabled. |
| 0b1 | When executing at any Exception level, and EL2 is enabled in the current Security state: <ul style="list-style-type: none"> • Physical SError interrupts are taken to EL2, unless they are routed to EL3. • When the value of HCR_EL2.TGE is 0, then virtual SError interrupts are enabled. |

If EL2 is enabled in the current Security state and the value of HCR_EL2.TGE is 1:

- Regardless of the value of the AMO bit physical asynchronous External aborts and SError interrupts target EL2 unless they are routed to EL3.
- When FEAT_VHE is not implemented, or if HCR_EL2.E2H is 0, this field behaves as 1 for all purposes other than a direct read of the value of this bit.
- When FEAT_VHE is implemented and HCR_EL2.E2H is 1, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

For more information, see 'Asynchronous exception routing'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMO, bit [4]

Physical IRQ Routing.

| IMO | Meaning |
|-----|--|
| 0b0 | When executing at Exception levels below EL2, and EL2 is enabled in the current Security state: <ul style="list-style-type: none"> When the value of HCR_EL2.TGE is 0, Physical IRQ interrupts are not taken to EL2. When the value of HCR_EL2.TGE is 1, Physical IRQ interrupts are taken to EL2 unless they are routed to EL3. Virtual IRQ interrupts are disabled. |
| 0b1 | When executing at any Exception level, and EL2 is enabled in the current Security state: <ul style="list-style-type: none"> Physical IRQ interrupts are taken to EL2, unless they are routed to EL3. When the value of HCR_EL2.TGE is 0, then Virtual IRQ interrupts are enabled. |

If EL2 is enabled in the current Security state, and the value of HCR_EL2.TGE is 1:

- Regardless of the value of the IMO bit, physical IRQ Interrupts target EL2 unless they are routed to EL3.
- When FEAT_VHE is not implemented, or if HCR_EL2.E2H is 0, this field behaves as 1 for all purposes other than a direct read of the value of this bit.
- When FEAT_VHE is implemented and HCR_EL2.E2H is 1, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

For more information, see 'Asynchronous exception routing'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FMO, bit [3]

Physical FIQ Routing.

| FMO | Meaning |
|-----|--|
| 0b0 | When executing at Exception levels below EL2, and EL2 is enabled in the current Security state: <ul style="list-style-type: none"> When the value of HCR_EL2.TGE is 0, Physical FIQ interrupts are not taken to EL2. When the value of HCR_EL2.TGE is 1, Physical FIQ interrupts are taken to EL2 unless they are routed to EL3. Virtual FIQ interrupts are disabled. |
| 0b1 | When executing at any Exception level, and EL2 is enabled in the current Security state: <ul style="list-style-type: none"> Physical FIQ interrupts are taken to EL2, unless they are routed to EL3. When HCR_EL2.TGE is 0, then Virtual FIQ interrupts are enabled. |

If EL2 is enabled in the current Security state and the value of HCR_EL2.TGE is 1:

- Regardless of the value of the FMO bit, physical FIQ Interrupts target EL2 unless they are routed to EL3.
- When FEAT_VHE is not implemented, or if HCR_EL2.E2H is 0, this field behaves as 1 for all purposes other than a direct read of the value of this bit.
- When FEAT_VHE is implemented and HCR_EL2.E2H is 1, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

For more information, see 'Asynchronous exception routing'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PTW, bit [2]

Protected Table Walk. In the EL1&0 translation regime, a translation table access made as part of a stage 1 translation table walk is subject to a stage 2 translation. The combining of the memory type attributes from the

two stages of translation means the access might be made to a type of Device memory. If this occurs, then the value of this bit determines the behavior:

| PTW | Meaning |
|-----|--|
| 0b0 | The translation table walk occurs as if it is to Normal Non-cacheable memory. This means it can be made speculatively. |
| 0b1 | The memory access generates a stage 2 Permission fault. |

This bit is permitted to be cached in a TLB.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SWIO, bit [1]

Set/Way Invalidation Override. Causes EL1 execution of the data cache invalidate by set/way instructions to perform a data cache clean and invalidate by set/way:

| SWIO | Meaning |
|------|---|
| 0b0 | This control has no effect on the operation of data cache invalidate by set/way instructions. |
| 0b1 | Data cache invalidate by set/way instructions perform a data cache clean and invalidate by set/way. |

When the value of this bit is 1:

AArch32: [DCISW](#) performs the same invalidation as a [DCCISW](#) instruction.

AArch64: [DC ISW](#) performs the same invalidation as a [DC CISW](#) instruction.

This bit can be implemented as RES1.

When HCR_EL2.TGE is 1, the PE ignores the value of this field for all purposes other than a direct read of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VM, bit [0]

Virtualization enable. Enables stage 2 address translation for the EL1&0 translation regime, when EL2 is enabled in the current Security state.

| VM | Meaning |
|-----|---|
| 0b0 | EL1&0 stage 2 address translation disabled. |
| 0b1 | EL1&0 stage 2 address translation enabled. |

When the value of this bit is 1, data cache invalidate instructions executed at EL1 perform a data cache clean and invalidate. For the invalidate by set/way instruction this behavior applies regardless of the value of the HCR_EL2.SWIO bit.

This bit is permitted to be cached in a TLB.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x078];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HCR_EL2;
elsif PSTATE.EL == EL3 then
    return HCR_EL2;

```

MSR HCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x078] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    HCR_EL2 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HCRX_EL2, Extended Hypervisor Configuration Register

The HCRX_EL2 characteristics are:

Purpose

Provides configuration controls for virtualization, including defining whether various operations are trapped to EL2.

Configuration

This register is present only when FEAT_HCX is implemented. Otherwise, direct accesses to HCRX_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

The bits in this register behave as if they are 0 for all purposes other than direct reads of the register if:

- EL2 is not enabled in the current Security state.
- [SCR_EL3.HXEn](#) is 0.

Attributes

HCRX_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | |
|--|-------|------|------|-------|-------|---------|-------|--------|------|-------|------|
| 6362616059585756555453525150494847464544 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 |
| RES0 | | | | | | | | | | | |
| RES0 | MSCEn | MCE2 | CMOW | VFNMI | VINMI | TALLINT | SMPME | FGTnXS | FnXS | EnASR | EnAL |
| 3130292827262524232221201918171615141312 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Bits [63:12]

Reserved, RES0.

MSCEn, bit [11]

When FEAT_MOPS is implemented:

Memory Set and Memory Copy instructions Enable. Enables execution of the CPY*, SETG*, SETP*, SETM*, and SETE* instructions at EL1 or EL0.

| MSCEn | Meaning |
|-------|--|
| 0b0 | Execution of the Memory Copy and Memory Set instructions is UNDEFINED at EL1 or EL0. |
| 0b1 | This control does not cause any instructions to be UNDEFINED. |

If EL2 is not implemented or enabled, this bit behaves as if it is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MCE2, bit [10]**When FEAT_MOPS is implemented:**

Controls Memory Copy and Memory Set exceptions generated as part of attempting to execute the Memory Copy and Memory Set instructions from EL1.

| MCE2 | Meaning |
|------|--|
| 0b0 | Memory Copy and Memory Set exceptions generated from EL1 are taken to EL1. |
| 0b1 | Memory Copy and Memory Set exceptions generated from EL1 are taken to EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CMOW, bit [9]**When FEAT_CMOW is implemented:**

Controls cache maintenance instruction permission for the following instructions executed at EL1 or EL0.

- [IC IVAU](#), [DC CIVAC](#), [DC CIGDVAC](#) and [DC CIGVAC](#).
- [ICIMVAU](#), [DCCIMVAC](#).

| CMOW | Meaning |
|------|--|
| 0b0 | These instructions executed at EL1 or EL0 with stage 2 read permission, but without stage 2 write permission do not generate a stage 2 permission fault. |
| 0b1 | These instructions executed at EL1 or EL0, if enabled as a result of SCTLR_EL1.UCI =1, with stage 2 read permission, but without stage 2 write permission generate a stage 2 permission fault. |

For this control, stage 2 has write permission if S2AP[1] is 1 or DBM is 1 in the stage 2 descriptor. The instructions do not cause an update to the dirty state.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VFNMI, bit [8]**When FEAT_NMI is implemented:**

Virtual FIQ Interrupt with Superpriority. Enables signaling of virtual FIQ interrupts with Superpriority.

| VFNMI | Meaning |
|-------|---|
| 0b0 | When HCR_EL2.VF is 1, a signaled pending virtual FIQ interrupt does not have Superpriority. |
| 0b1 | When HCR_EL2.VF is 1, a signaled pending virtual FIQ interrupt has Superpriority. |

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0.

Otherwise:

Reserved, RES0.

VINMI, bit [7]

When FEAT_NMI is implemented:

Virtual IRQ Interrupt with Superpriority. Enables signaling of virtual IRQ interrupts with Superpriority.

| VINMI | Meaning |
|-------|---|
| 0b0 | When HCR_EL2.VI is 1, a signaled pending virtual IRQ interrupt does not have Superpriority. |
| 0b1 | When HCR_EL2.VI is 1, a signaled pending virtual IRQ interrupt has Superpriority. |

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0.

Otherwise:

Reserved, RES0.

TALLINT, bit [6]

When FEAT_NMI is implemented:

Trap MSR writes of [ALLINT](#) at EL1 using AArch64 to EL2, when EL2 is implemented and enabled in the current Security state, reported using EC syndrome value 0x18.

| TALLINT | Meaning |
|---------|---|
| 0b0 | MSR writes of ALLINT are not trapped by this mechanism. |
| 0b1 | MSR writes of ALLINT at EL1 using AArch64 are trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0.

Otherwise:

Reserved, RES0.

SMPME, bit [5]

When FEAT_SME is implemented:

Streaming Mode Priority Mapping Enable.

Controls mapping of the value of [SMPRI_EL1](#). Priority for streaming execution priority at EL0 or EL1.

| SMPME | Meaning |
|-------|---|
| 0b0 | The effective priority value is taken from SMPRI_EL1.Priority . |
| 0b1 | The effective priority value is: <ul style="list-style-type: none"> When the current Exception level is EL2 or EL3, the value of SMPRI_EL1.Priority. When the current Exception level is EL0 or EL1, the value of the SMPRMAP_EL2 field corresponding to the value of SMPRI_EL1.Priority. |

When [SMIDR_EL1.SMPS](#) is '0', this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FGTnXS, bit [4]

When FEAT_XS is implemented:

Determines if the fine-grained traps in HFGITR_EL2 that apply to each of the TLBI maintenance instructions that are accessible at EL1 also apply to the corresponding TLBI maintenance instructions with the nXS qualifier.

| FGTnXS | Meaning |
|--------|---|
| 0b0 | The fine-grained trap in the HFGITR_EL2 that applies to a TLBI maintenance instruction at EL1 also applies to the corresponding TLBI instruction with the nXS qualifier at EL1. |
| 0b1 | The fine-grained trap in the HFGITR_EL2 that applies to a TLBI maintenance instruction at EL1 does not apply to the corresponding TLBI instruction with the nXS qualifier at EL1. |

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0.

Otherwise:

Reserved, RES0.

FnXS, bit [3]

When FEAT_XS is implemented:

Determines the behavior of TLBI instructions affected by the XS attribute.

This control bit also determines whether an AArch64 DSB instruction behaves as a DSB instruction with an nXS qualifier when executed at EL0 and EL1.

| FnXS | Meaning |
|------|---|
| 0b0 | This control does not have any effect on the behavior of the TLBI maintenance instructions. |
| 0b1 | A TLBI maintenance instruction without the nXS qualifier executed at EL1 behaves in the same way as the corresponding TLBI maintenance instruction with the nXS qualifier. An AArch64 DSB instruction executed at EL1 or EL0 behaves in the same way as the corresponding DSB instruction with the nXS qualifier executed at EL1 or EL0. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0.

Otherwise:

Reserved, RES0.

EnASR, bit [2]

When FEAT_LS64 is implemented:

When [HCR_EL2](#).{E2H, TGE} != {1, 1}, traps execution of an ST64BV instruction at EL0 or EL1 to EL2.

| EnASR | Meaning |
|-------|---|
| 0b0 | Execution of an ST64BV instruction at EL0 is trapped to EL2 if the execution is not trapped by SCTLR_EL1 .EnASR. Execution of an ST64BV instruction at EL1 is trapped to EL2. |
| 0b1 | This control does not cause any instructions to be trapped. |

A trap of an ST64BV instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000000.

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0.

Otherwise:

Reserved, RES0.

EnALS, bit [1]

When FEAT_LS64 is implemented:

When [HCR_EL2](#).{E2H, TGE} != {1, 1}, traps execution of an LD64B or ST64B instruction at EL0 or EL1 to EL2.

| EnALS | Meaning |
|-------|---|
| 0b0 | Execution of an LD64B or ST64B instruction at EL0 is trapped to EL2 if the execution is not trapped by SCTLR_EL1 .EnALS. Execution of an LD64B or ST64B instruction at EL1 is trapped to EL2. |
| 0b1 | This control does not cause any instructions to be trapped. |

A trap of an LD64B or ST64B instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000002.

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0.

Otherwise:

Reserved, RES0.

EnAS0, bit [0]

When FEAT_LS64 is implemented:

When [HCR_EL2](#).{E2H, TGE} != {1, 1}, traps execution of an ST64BV0 instruction at EL0 or EL1 to EL2.

| EnAS0 | Meaning |
|-------|---|
| 0b0 | Execution of an ST64BV0 instruction at EL0 is trapped to EL2 if the execution is not trapped by SCTLR_EL1 .EnAS0. Execution of an ST64BV0 instruction at EL1 is trapped to EL2. |
| 0b1 | This control does not cause any instructions to be trapped. |

A trap of an ST64BV0 instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, when EL3 is not implemented and EL2 is implemented, this field resets to 0.

Otherwise:

Reserved, RES0.

Accessing HCRX_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HCRX_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0xA0];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.HXEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.HXEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return HCRX_EL2;
elsif PSTATE.EL == EL3 then
    return HCRX_EL2;

```

MSR HCRX_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0xA0] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.HXEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.HXEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HCRX_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    HCRX_EL2 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HDFGRTR_EL2, Hypervisor Debug Fine-Grained Read Trap Register

The HDFGRTR_EL2 characteristics are:

Purpose

Provides controls for traps of MRS and MRC reads of debug, trace, PMU, and Statistical Profiling System registers.

Configuration

This register is present only when FEAT_FGT is implemented. Otherwise, direct accesses to HDFGRTR_EL2 are UNDEFINED.

Attributes

HDFGRTR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | |
|----------------------------|-------------------------------|----------------------------|----------------------------|-----------------------------|-----------------------------|-------------------------------|----------------------------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 |
| PMBIDR_EL1 | nPMSNEVFR_EL1 | nBRBDATA | nBRBCTL | nBRBIDR | PMCEIDn_EL0 | PMUSERENR_EL0 | TRBTRG_EL1 |
| PMSIRR_EL1 | PMSIDR_EL1 | PMSICR_EL1 | PMSFCR_EL1 | PMSEVFR_EL1 | PMSCR_EL1 | PMBSR_EL1 | PMBPTR_EL1 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |

PMBIDR_EL1, bit [63]

When FEAT_SPE is implemented:

Trap MRS reads of [PMBIDR_EL1](#) at EL1 using AArch64 to EL2.

| PMBIDR_EL1 | Meaning |
|------------|---|
| 0b0 | MRS reads of PMBIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMBIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

nPMSNEVFR_EL1, bit [62]

When FEAT_SPEv1p2 is implemented:

Trap MRS reads of [PMSNEVFR_EL1](#) at EL1 using AArch64 to EL2.

| nPMSNEVFR_EL1 | Meaning |
|----------------------|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSNEVFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of PMSNEVFR_EL1 are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

nBRBDATA, bit [61]

When FEAT_BRBE is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [BRBINF<n>_EL1](#).
- [BRBINFINJ_EL1](#).
- [BRBSRC<n>_EL1](#).
- [BRBSRCINJ_EL1](#).
- [BRBTGT<n>_EL1](#).
- [BRBTGTINJ_EL1](#).
- [BRBTS_EL1](#).

| nBRBDATA | Meaning |
|-----------------|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the System registers listed above are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

nBRBCTL, bit [60]

When FEAT_BRBE is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [BRBCR_EL1](#).
- [BRBFCR_EL1](#).

| nBRBCTL | Meaning |
|---------|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of the System registers listed above are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

nBRBIDR, bit [59]

When FEAT_BRBE is implemented:

Trap MRS reads of [BRBIDR0_EL1](#) at EL1 using AArch64 to EL2.

| nBRBIDR | Meaning |
|---------|--|
| 0b0 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of BRBIDR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of BRBIDR0_EL1 are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMCEIDn_EL0, bit [58]

When FEAT_PMUv3 is implemented:

Trap MRS reads of PMCEID<n>_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMCEID<n> at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| PMCEIDn_EL0 | Meaning |
|-------------|---|
| 0b0 | MRS reads of PMCEID<n>_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMCEID<n> at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> MRS reads of PMCEID<n>_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MRC reads of PMCEID<n> at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMUSERENR_EL0, bit [57]

When FEAT_PMUv3 is implemented:

Trap MRS reads of [PMUSERENR_EL0](#) at EL1 and EL0 using AArch64 and MRC reads of [PMUSERENR](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| PMUSERENR_EL0 | Meaning |
|---------------|---|
| 0b0 | MRS reads of PMUSERENR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMUSERENR at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> MRS reads of PMUSERENR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MRC reads of PMUSERENR at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBTRG_EL1, bit [56]

When FEAT_TRBE is implemented:

Trap MRS reads of [TRBTRG_EL1](#) at EL1 using AArch64 to EL2.

| TRBTRG_EL1 | Meaning |
|------------|---|
| 0b0 | MRS reads of TRBTRG_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBTRG_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBSR_EL1, bit [55]

When FEAT_TRBE is implemented:

Trap MRS reads of [TRBSR_EL1](#) at EL1 using AArch64 to EL2.

| TRBSR_EL1 | Meaning |
|-----------|--|
| 0b0 | MRS reads of TRBSR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBPTR_EL1, bit [54]

When FEAT_TRBE is implemented:

Trap MRS reads of [TRBPTR_EL1](#) at EL1 using AArch64 to EL2.

| TRBPTR_EL1 | Meaning |
|------------|---|
| 0b0 | MRS reads of TRBPTR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBPTR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBMAR_EL1, bit [53]

When FEAT_TRBE is implemented:

Trap MRS reads of [TRBMAR_EL1](#) at EL1 using AArch64 to EL2.

| TRBMAR_EL1 | Meaning |
|------------|---|
| 0b0 | MRS reads of TRBMAR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBMAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBLIMITR_EL1, bit [52]

When FEAT_TRBE is implemented:

Trap MRS reads of [TRBLIMITR_EL1](#) at EL1 using AArch64 to EL2.

| TRBLIMITR_EL1 | Meaning |
|---------------|--|
| 0b0 | MRS reads of TRBLIMITR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBLIMITR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBIDR_EL1, bit [51]

When FEAT_TRBE is implemented:

Trap MRS reads of [TRBIDR_EL1](#) at EL1 using AArch64 to EL2.

| TRBIDR_EL1 | Meaning |
|------------|---|
| 0b0 | MRS reads of TRBIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBBASER_EL1, bit [50]

When FEAT_TRBE is implemented:

Trap MRS reads of [TRBBASER_EL1](#) at EL1 using AArch64 to EL2.

| TRBBASER_EL1 | Meaning |
|--------------|---|
| 0b0 | MRS reads of TRBBASER_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRBBASER_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [49]

Reserved, RES0.

TRCVICTLR, bit [48]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of [TRCVICTLR](#) at EL1 using AArch64 to EL2.

| TRCVICTLR | Meaning |
|-----------|--|
| 0b0 | MRS reads of TRCVICTLR are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCVICTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCSTATR, bit [47]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of [TRCSTATR](#) at EL1 using AArch64 to EL2.

| TRCSTATR | Meaning |
|----------|---|
| 0b0 | MRS reads of TRCSTATR are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCSTATR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCSSCSRn, bit [46]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented), TRCSSCSR<n> are implemented and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of [TRCSSCSR<n>](#) at EL1 using AArch64 to EL2.

| TRCSSCSRn | Meaning |
|-----------|--|
| 0b0 | MRS reads of TRCSSCSR<n> are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCSSCSR<n> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

If Single-shot Comparator n is not implemented, a read of [TRCSSCSR<n>](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCSEQSTR, bit [45]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented), TRCSEQSTR is implemented and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of [TRCSEQSTR](#) at EL1 using AArch64 to EL2.

| TRCSEQSTR | Meaning |
|-----------|--|
| 0b0 | MRS reads of TRCSEQSTR are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCSEQSTR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCPRGCTLR, bit [44]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of [TRCPRGCTLR](#) at EL1 using AArch64 to EL2.

| TRCPRGCTLR | Meaning |
|------------|---|
| 0b0 | MRS reads of TRCPRGCTLR are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCPRGCTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCOSLSR, bit [43]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of [TRCOSLSR](#) at EL1 using AArch64 to EL2.

| TRCOSLSR | Meaning |
|----------|---|
| 0b0 | MRS reads of TRCOSLSR are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCOSLSR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [42]

Reserved, RES0.

TRCIMSPECn, bit [41]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of [TRCIMSPEC<n>](#) at EL1 using AArch64 to EL2.

| TRCIMSPECn | Meaning |
|------------|---|
| 0b0 | MRS reads of TRCIMSPEC<n> are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCIMSPEC<n> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

TRCIMSPEC<1-7> are optional. If [TRCIMSPEC<n>](#) is not implemented, a read of [TRCIMSPEC<n>](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCID, bit [40]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [TRCDEVARCH](#).
- [TRCDEVID](#).
- [TRCIDR<n>](#).

| TRCID | Meaning |
|-------|---|
| 0b0 | MRS reads of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [39:38]

Reserved, RES0.

TRCCNTVRn, bit [37]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented), TRCCNTVR<n> are implemented and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of [TRCCNTVR<n>](#) at EL1 using AArch64 to EL2.

| TRCCNTVRn | Meaning |
|-----------|--|
| 0b0 | MRS reads of TRCCNTVR<n> are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCCNTVR<n> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

If Counter n is not implemented, a read of [TRCCNTVR<n>](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCCCLAIM, bit [36]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [TRCCCLAIMCLR](#).
- [TRCCCLAIMSET](#).

| TRCCCLAIM | Meaning |
|-----------|---|
| 0b0 | MRS reads of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCAUXCTLR, bit [35]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of [TRCAUXCTLR](#) at EL1 using AArch64 to EL2.

| TRCAUXCTLR | Meaning |
|------------|---|
| 0b0 | MRS reads of TRCAUXCTLR are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCAUXCTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCAUTHSTATUS, bit [34]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of [TRCAUTHSTATUS](#) at EL1 using AArch64 to EL2.

| TRCAUTHSTATUS | Meaning |
|---------------|--|
| 0b0 | MRS reads of TRCAUTHSTATUS are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TRCAUTHSTATUS at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRC, bit [33]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [TRCACATR<n>](#).
- [TRCACVR<n>](#).
- [TRCBBCTLR](#).
- [TRCCCCTLR](#).
- [TRCCIDCCTLRO](#).
- [TRCCIDCCTLRI](#).
- [TRCCIDCVR<n>](#).

- [TRCCNTCTLR<n>](#).
- [TRCCNTRLDVR<n>](#).
- [TRCCONFIGR](#).
- [TRCEVENTCTL0R](#).
- [TRCEVENTCTL1R](#).
- [TRCEXTINSELR](#), if FEAT_ETMv4 is implemented.
- [TRCEXTINSELR<n>](#), if FEAT_ETE is implemented.
- [TRCQCTLR](#).
- [TRCRSCTLR<n>](#).
- [TRCRSR](#), if FEAT_ETE is implemented.
- [TRCSEQEVR<n>](#).
- [TRCSEQRSTEV](#).
- [TRCSSCCR<n>](#).
- [TRCSSPCICR<n>](#).
- [TRCSTALLCTLR](#).
- [TRCSYNCP](#).
- [TRCTRACEIDR](#).
- [TRCTSCTLR](#).
- [TRCVIIECTLR](#).
- [TRCVIPCSSCTLR](#).
- [TRCVISSCTLR](#).
- [TRCVMIDCCTLR0](#).
- [TRCVMIDCCTLR1](#).
- [TRCVMIDCVR<n>](#).

| TRC | Meaning |
|-----|---|
| 0b0 | MRS reads of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

A read of an unimplemented register is UNDEFINED.

[TRCEXTINSELR<n>](#) and [TRCRSR](#) are only implemented if FEAT_ETE is implemented.

TRCEXTINSELR is only implemented if FEAT_ETE is not implemented and FEAT_ETMv4 is implemented.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSLATFR_EL1, bit [32]

When FEAT_SPE is implemented:

Trap MRS reads of [PMSLATFR_EL1](#) at EL1 using AArch64 to EL2.

| PMSLATFR_EL1 | Meaning |
|--------------|---|
| 0b0 | MRS reads of PMSLATFR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSLATFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSIRR_EL1, bit [31]

When FEAT_SPE is implemented:

Trap MRS reads of [PMSIRR_EL1](#) at EL1 using AArch64 to EL2.

| PMSIRR_EL1 | Meaning |
|-------------------|---|
| 0b0 | MRS reads of PMSIRR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSIRR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSIDR_EL1, bit [30]

When FEAT_SPE is implemented:

Trap MRS reads of [PMSIDR_EL1](#) at EL1 using AArch64 to EL2.

| PMSIDR_EL1 | Meaning |
|-------------------|---|
| 0b0 | MRS reads of PMSIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSICR_EL1, bit [29]

When FEAT_SPE is implemented:

Trap MRS reads of [PMSICR_EL1](#) at EL1 using AArch64 to EL2.

| PMSICR_EL1 | Meaning |
|-------------------|---|
| 0b0 | MRS reads of PMSICR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSICR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSFCR_EL1, bit [28]

When FEAT_SPE is implemented:

Trap MRS reads of [PMSFCR_EL1](#) at EL1 using AArch64 to EL2.

| PMSFCR_EL1 | Meaning |
|-------------------|---|
| 0b0 | MRS reads of PMSFCR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSFCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSEVFR_EL1, bit [27]

When FEAT_SPE is implemented:

Trap MRS reads of [PMSEVFR_EL1](#) at EL1 using AArch64 to EL2.

| PMSEVFR_EL1 | Meaning |
|--------------------|--|
| 0b0 | MRS reads of PMSEVFR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSEVFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSCR_EL1, bit [26]

When FEAT_SPE is implemented:

Trap MRS reads of [PMSCR_EL1](#) at EL1 using AArch64 to EL2.

| PMSCR_EL1 | Meaning |
|-----------|--|
| 0b0 | MRS reads of PMSCR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMSCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMBSR_EL1, bit [25]

When FEAT_SPE is implemented:

Trap MRS reads of [PMBSR_EL1](#) at EL1 using AArch64 to EL2.

| PMBSR_EL1 | Meaning |
|-----------|--|
| 0b0 | MRS reads of PMBSR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMBSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMBPTR_EL1, bit [24]

When FEAT_SPE is implemented:

Trap MRS reads of [PMBPTR_EL1](#) at EL1 using AArch64 to EL2.

| PMBPTR_EL1 | Meaning |
|-------------------|---|
| 0b0 | MRS reads of PMBPTR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMBPTR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMBLIMITR_EL1, bit [23]

When FEAT_SPE is implemented:

Trap MRS reads of [PMBLIMITR_EL1](#) at EL1 using AArch64 to EL2.

| PMBLIMITR_EL1 | Meaning |
|----------------------|--|
| 0b0 | MRS reads of PMBLIMITR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMBLIMITR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMMIR_EL1, bit [22]

When FEAT_PMUv3 is implemented:

Trap MRS reads of [PMMIR_EL1](#) at EL1 using AArch64 to EL2.

| PMMIR_EL1 | Meaning |
|------------------|--|
| 0b0 | MRS reads of PMMIR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PMMIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [21:20]

Reserved, RES0.

PMSELR_EL0, bit [19]**When FEAT_PMUv3 is implemented:**

Trap MRS reads of [PMSELR_EL0](#) at EL1 and EL0 using AArch64 and MRC reads of [PMSELR](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| PMSELR_EL0 | Meaning |
|------------|---|
| 0b0 | MRS reads of PMSELR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMSELR at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> MRS reads of PMSELR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MRC reads of PMSELR at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMOVS, bit [18]**When FEAT_PMUv3 is implemented:**

Trap MRS reads and MRC reads of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MRS reads of [PMOVSLR_EL0](#) and [PMOVSSET_EL0](#).
- At EL0 using AArch32 when EL1 is using AArch64: MRC reads of [PMOVS](#) and [PMOVSSET](#).

| PMOVS | Meaning |
|-------|---|
| 0b0 | The operations listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> MRS reads at EL1 and EL0 using AArch64 of PMOVSLR_EL0 and PMOVSSET_EL0 are trapped to EL2 and reported with EC syndrome value 0x18. MRC reads at EL0 using AArch32 of PMOVS and PMOVSSET are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMINTEN, bit [17]

When FEAT_PMUv3 is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [PMINTENCLR_EL1](#).
- [PMINTENSET_EL1](#).

| PMINTEN | Meaning |
|---------|---|
| 0b0 | MRS reads of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMCNTEN, bit [16]

When FEAT_PMUv3 is implemented:

Trap MRS reads and MRC reads of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MRS reads of [PMCNTENCLR_EL0](#) and [PMCNTENSET_EL0](#).
- At EL0 using AArch32 when EL1 is using AArch64: MRC reads of [PMCNTENCLR](#) and [PMCNTENSET](#).

| PMCNTEN | Meaning |
|---------|---|
| 0b0 | The operations listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> MRS reads at EL1 and EL0 using AArch64 of PMCNTENCLR_EL0 and PMCNTENSET_EL0 are trapped to EL2 and reported with EC syndrome value 0x18. MRC reads at EL0 using AArch32 of PMCNTENCLR and PMCNTENSET are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMCCNTR_EL0, bit [15]**When FEAT_PMUv3 is implemented:**

Trap MRS reads of [PMCCNTR_EL0](#) at EL1 and EL0 using AArch64 and MRC and MRRC reads of [PMCCNTR](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| PMCCNTR_EL0 | Meaning |
|--------------------|---|
| 0b0 | MRS reads of PMCCNTR_EL0 at EL1 and EL0 using AArch64 and MRC and MRRC reads of PMCCNTR at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> MRS reads of PMCCNTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MRC and MRRC reads of PMCCNTR at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03 (for MRC) or 0x04 (for MRRC). |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMCCFILTR_EL0, bit [14]**When FEAT_PMUv3 is implemented:**

Trap MRS reads of [PMCCFILTR_EL0](#) at EL1 and EL0 using AArch64 and MRC reads of [PMCCFILTR](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| PMCCFILTR_EL0 | Meaning |
|----------------------|---|
| 0b0 | MRS reads of PMCCFILTR_EL0 at EL1 and EL0 using AArch64 and MRC reads of PMCCFILTR at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> MRS reads of PMCCFILTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MRC reads of PMCCFILTR at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

[PMCCFILTR_EL0](#) can also be accessed in AArch64 state using [PMXEVTYPER_EL0](#) when [PMSELR_EL0](#).SEL == 31, and [PMCCFILTR](#) can also be accessed in AArch32 state using [PMXEVTYPER](#) when [PMSELR](#).SEL == 31.

Setting this field to 1 has no effect on accesses to [PMEVTYPER_EL0](#) and [PMXEVTYPER](#), regardless of the value of [PMSELR_EL0.SEL](#) or [PMSELR.SEL](#).

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMEVTYPERn_EL0, bit [13]

When FEAT_PMuV3 is implemented:

Trap MRS reads and MRC reads of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MRS reads of [PMEVTYPER<n>_EL0](#) and [PMXEVTYPER_EL0](#).
- At EL0 using AArch32 when EL1 is using AArch64: MRC reads of [PMEVTYPER<n>](#) and [PMXEVTYPER](#).

| PMEVTYPERn_EL0 | Meaning |
|----------------|---|
| 0b0 | The operations listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> MRS reads at EL1 and EL0 using AArch64 of PMEVTYPER<n>_EL0 and PMXEVTYPER_EL0 are trapped to EL2 and reported with EC syndrome value 0x18. MRC reads at EL0 using AArch32 of PMEVTYPER<n> and PMXEVTYPER are trapped to EL2 and reported with EC syndrome value 0x03. |

Regardless of the value of this field, for each value n:

- If event counter n is not implemented, the following accesses are UNDEFINED:
 - In AArch64 state, a read of [PMEVTYPER<n>_EL0](#), or, if n is not 31, a read of [PMXEVTYPER_EL0](#) when [PMSELR_EL0.SEL](#) == n.
 - In AArch32 state, a read of [PMEVTYPER<n>](#), or, if n is not 31, a read of [PMXEVTYPER](#) when [PMSELR.SEL](#) == n.
- If event counter n is implemented, n is greater-than-or-equal-to [MDCR_EL2.HPMN](#), and EL2 is implemented and enabled in the current Security state, the following generate a Trap exception to EL2 from EL0 or EL1:
 - In AArch64 state, a read of [PMEVTYPER<n>_EL0](#), or a read of [PMXEVTYPER_EL0](#) when [PMSELR_EL0.SEL](#) == n, reported with EC syndrome value 0x18.
 - In AArch32 state, a read of [PMEVTYPER<n>](#), or a read of [PMXEVTYPER](#) when [PMSELR.SEL](#) == n, reported with EC syndrome value 0x03.

See also [HDFGRTR_EL2.PMCCFILTR_EL0](#).

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMEVCNTRn_EL0, bit [12]**When FEAT_PMUv3 is implemented:**

Trap MRS reads and MRC reads of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MRS reads of [PMEVCNTR<n>_EL0](#) and [PMXEVNTR_EL0](#).
- At EL0 using AArch32 when EL1 is using AArch64: MRC reads of [PMEVCNTR<n>](#) and [PMXEVNTR](#).

| PMEVCNTRn_EL0 | Meaning |
|----------------------|---|
| 0b0 | The operations listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> • MRS reads at EL1 and EL0 using AArch64 of PMEVCNTR<n>_EL0 and PMXEVNTR_EL0 are trapped to EL2 and reported with EC syndrome value 0x18. • MRC reads at EL0 using AArch32 of PMEVCNTR<n> and PMXEVNTR are trapped to EL2 and reported with EC syndrome value 0x03. |

Regardless of the value of this field, for each value n:

- If event counter n is not implemented, the following accesses are UNDEFINED:
 - In AArch64 state, a read of [PMEVCNTR<n>_EL0](#), or a read of [PMXEVNTR_EL0](#) when [PMSELR_EL0](#).SEL == n.
 - In AArch32 state, a read of [PMEVCNTR<n>](#), or a read of [PMXEVNTR](#) when [PMSELR](#).SEL == n.
- If event counter n is implemented, n is greater-than-or-equal-to [MDCR_EL2](#).HPMN, and EL2 is implemented and enabled in the current Security state, the following generate a Trap exception to EL2 from EL0 or EL1:
 - In AArch64 state, a read of [PMEVCNTR<n>_EL0](#), or a read of [PMXEVNTR_EL0](#) when [PMSELR_EL0](#).SEL == n, reported with EC syndrome value 0x18.
 - In AArch32 state, a read of [PMEVCNTR<n>](#), or a read of [PMXEVNTR](#) when [PMSELR](#).SEL == n, reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

OSDLR_EL1, bit [11]**When FEAT_DoubleLock is implemented:**

Trap MRS reads of [OSDLR_EL1](#) at EL1 using AArch64 to EL2.

| OSDLR_EL1 | Meaning |
|------------------|---|
| 0b0 | MRS reads of OSDLR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then MRS reads of OSDLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

OSECCR_EL1, bit [10]

Trap MRS reads of [OSECCR_EL1](#) at EL1 using AArch64 to EL2.

| OSECCR_EL1 | Meaning |
|-------------------|---|
| 0b0 | MRS reads of OSECCR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of OSECCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

OSLSR_EL1, bit [9]

Trap MRS reads of [OSLSR_EL1](#) at EL1 using AArch64 to EL2.

| OSLSR_EL1 | Meaning |
|------------------|--|
| 0b0 | MRS reads of OSLSR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of OSLSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Bit [8]

Reserved, RES0.

DBGPRCR_EL1, bit [7]

Trap MRS reads of [DBGPRCR_EL1](#) at EL1 using AArch64 to EL2.

| DBGPRCR_EL1 | Meaning |
|--------------------|--|
| 0b0 | MRS reads of DBGPRCR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DBGPRCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DBGAUTHSTATUS_EL1, bit [6]

Trap MRS reads of [DBGAUTHSTATUS_EL1](#) at EL1 using AArch64 to EL2.

| DBGAUTHSTATUS_EL1 | Meaning |
|-------------------|--|
| 0b0 | MRS reads of DBGAUTHSTATUS_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DBGAUTHSTATUS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DBGCLAIM, bit [5]

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [DBGCLAIMCLR_EL1](#).
- [DBGCLAIMSET_EL1](#).

| DBGCLAIM | Meaning |
|----------|---|
| 0b0 | MRS reads of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

MDSCR_EL1, bit [4]

Trap MRS reads of [MDSCR_EL1](#) at EL1 using AArch64 to EL2.

| MDSCR_EL1 | Meaning |
|-----------|--|
| 0b0 | MRS reads of MDSCR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of MDSCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DBGWVRn_EL1, bit [3]

Trap MRS reads of [DBGWVR<n>_EL1](#) at EL1 using AArch64 to EL2.

| DBGWVRn_EL1 | Meaning |
|-------------|--|
| 0b0 | MRS reads of DBGWVR<n>_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DBGWVR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

If watchpoint n is not implemented, a read of [DBGWVR<n>_EL1](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DBGWCRn_EL1, bit [2]

Trap MRS reads of [DBGWCR<n>_EL1](#) at EL1 using AArch64 to EL2.

| DBGWCRn_EL1 | Meaning |
|-------------|--|
| 0b0 | MRS reads of DBGWCR<n>_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DBGWCR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

If watchpoint n is not implemented, a read of [DBGWCR<n>_EL1](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DBGBVRn_EL1, bit [1]

Trap MRS reads of [DBGBVR<n>_EL1](#) at EL1 using AArch64 to EL2.

| DBGBVRn_EL1 | Meaning |
|-------------|--|
| 0b0 | MRS reads of DBGBVR<n>_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DBGBVR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

If breakpoint n is not implemented, a read of [DBGBVR<n>_EL1](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DBGBCRn_EL1, bit [0]

Trap MRS reads of [DBGBCR<n>_EL1](#) at EL1 using AArch64 to EL2.

| DBGBCRn_EL1 | Meaning |
|-------------|--|
| 0b0 | MRS reads of DBGBCR<n>_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DBGBCR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

If breakpoint n is not implemented, a read of [DBGBCR<n>_EL1](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Accessing HDFGRTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HDFGRTR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0011 | 0b0001 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x1D0];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return HDFGRTR_EL2;
elsif PSTATE.EL == EL3 then
    return HDFGRTR_EL2;

```

MSR HDFGRTR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0011 | 0b0001 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x1D0] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HDFGRTR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    HDFGRTR_EL2 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HDFGWTR_EL2, Hypervisor Debug Fine-Grained Write Trap Register

The HDFGWTR_EL2 characteristics are:

Purpose

Provides controls for traps of MSR and MCR writes of debug, trace, PMU, and Statistical Profiling System registers.

Configuration

This register is present only when FEAT_FGT is implemented. Otherwise, direct accesses to HDFGWTR_EL2 are UNDEFINED.

Attributes

HDFGWTR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | |
|------------|---------------|------------|------------|-------------|---------------|------------|------------|-----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | |
| RES0 | nPMSNEVFR_EL1 | nBRBDATA | nBRBCTL | RES0 | PMUSERENR_ELO | TRBTRG_EL1 | TF | |
| PMSIRR_EL1 | RES0 | PMSICR_EL1 | PMSFCR_EL1 | PMSEVFR_EL1 | PMSCR_EL1 | PMBSR_EL1 | PMBPTR_EL1 | PMB |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | |

Bit [63]

Reserved, RES0.

nPMSNEVFR_EL1, bit [62]

When FEAT_SPEv1p2 is implemented:

Trap MSR writes of [PMSNEVFR_EL1](#) at EL1 using AArch64 to EL2.

| nPMSNEVFR_EL1 | Meaning |
|---------------|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMSNEVFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |
| 0b1 | MSR writes of PMSNEVFR_EL1 are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

nBRBDATA, bit [61]**When FEAT_BRBE is implemented:**

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [BRBINFINJ_EL1](#).
- [BRBSRCINJ_EL1](#).
- [BRBTGTINJ_EL1](#).
- [BRBTS_EL1](#).

| nBRBDATA | Meaning |
|----------|--|
| 0b0 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTE _n == 1, then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |
| 0b1 | MSR writes of the System registers listed above are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

nBRBCTL, bit [60]**When FEAT_BRBE is implemented:**

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [BRBCR_EL1](#).
- [BRBFCCR_EL1](#).

| nBRBCTL | Meaning |
|---------|--|
| 0b0 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTE _n == 1, then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |
| 0b1 | MSR writes of the System registers listed above are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [59:58]

Reserved, RES0.

PMUSERENR_EL0, bit [57]**When FEAT_PMUv3 is implemented:**

Trap MSR writes of [PMUSERENR_EL0](#) at EL1 using AArch64 to EL2.

| PMUSERENR_EL0 | Meaning |
|----------------------|--|
| 0b0 | MSR writes of PMUSERENR_EL0 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMUSERENR_EL0 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBTRG_EL1, bit [56]**When FEAT_TRBE is implemented:**

Trap MSR writes of [TRBTRG_EL1](#) at EL1 using AArch64 to EL2.

| TRBTRG_EL1 | Meaning |
|-------------------|---|
| 0b0 | MSR writes of TRBTRG_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRBTRG_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBSR_EL1, bit [55]**When FEAT_TRBE is implemented:**

Trap MSR writes of [TRBSR_EL1](#) at EL1 using AArch64 to EL2.

| TRBSR_EL1 | Meaning |
|------------------|--|
| 0b0 | MSR writes of TRBSR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRBSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBPTR_EL1, bit [54]

When FEAT_TRBE is implemented:

Trap MSR writes of [TRBPTR_EL1](#) at EL1 using AArch64 to EL2.

| TRBPTR_EL1 | Meaning |
|------------|---|
| 0b0 | MSR writes of TRBPTR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRBPTR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBMAR_EL1, bit [53]

When FEAT_TRBE is implemented:

Trap MSR writes of [TRBMAR_EL1](#) at EL1 using AArch64 to EL2.

| TRBMAR_EL1 | Meaning |
|------------|---|
| 0b0 | MSR writes of TRBMAR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRBMAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRBLIMITR_EL1, bit [52]

When FEAT_TRBE is implemented:

Trap MSR writes of [TRBLIMITR_EL1](#) at EL1 using AArch64 to EL2.

| TRBLIMITR_EL1 | Meaning |
|---------------|--|
| 0b0 | MSR writes of TRBLIMITR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRBLIMITR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [51]

Reserved, RES0.

TRBBASER_EL1, bit [50]

When FEAT_TRBE is implemented:

Trap MSR writes of [TRBBASER_EL1](#) at EL1 using AArch64 to EL2.

| TRBBASER_EL1 | Meaning |
|--------------|---|
| 0b0 | MSR writes of TRBBASER_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRBBASER_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRFCR_EL1, bit [49]

When FEAT_TRF is implemented:

Trap MSR writes of [TRFCR_EL1](#) at EL1 using AArch64 to EL2.

| TRFCR_EL1 | Meaning |
|-----------|--|
| 0b0 | MSR writes of TRFCR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRFCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCVICTLR, bit [48]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MSR writes of [TRCVICTLR](#) at EL1 using AArch64 to EL2.

| TRCVICTLR | Meaning |
|-----------|--|
| 0b0 | MSR writes of TRCVICTLR are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCVICTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [47]

Reserved, RES0.

TRCSSCSRn, bit [46]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented), TRCSSCSR<n> are implemented and System register access to the PE Trace Unit registers is implemented:

Trap MSR writes of [TRCSSCSR<n>](#) at EL1 using AArch64 to EL2.

| TRCSSCSRn | Meaning |
|-----------|--|
| 0b0 | MSR writes of TRCSSCSR<n> are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCSSCSR<n> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

If Single-shot Comparator n is not implemented, a write of [TRCSSCSR<n>](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCSEQSTR, bit [45]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented), TRCSEQSTR is implemented and System register access to the PE Trace Unit registers is implemented:

Trap MSR writes of [TRCSEQSTR](#) at EL1 using AArch64 to EL2.

| TRCSEQSTR | Meaning |
|-----------|--|
| 0b0 | MSR writes of TRCSEQSTR are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCSEQSTR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCPRGCTLR, bit [44]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MSR writes of [TRCPRGCTLR](#) at EL1 using AArch64 to EL2.

| TRCPRGCTLR | Meaning |
|------------|---|
| 0b0 | MSR writes of TRCPRGCTLR are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCPRGCTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [43]

Reserved, RES0.

TRCOSLAR, bit [42]

When System register access to the PE Trace Unit registers is implemented and FEAT_ETMv4 is implemented:

Trap MSR writes of TRCOSLAR at EL1 using AArch64 to EL2.

| TRCOSLAR | Meaning |
|----------|---|
| 0b0 | MSR writes of TRCOSLAR are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCOSLAR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCIMSPECn, bit [41]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MSR writes of [TRCIMSPEC<n>](#) at EL1 using AArch64 to EL2.

| TRCIMSPECn | Meaning |
|------------|---|
| 0b0 | MSR writes of TRCIMSPEC<n> are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCIMSPEC<n> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

TRCIMSPEC<1-7> are optional. If [TRCIMSPEC<n>](#) is not implemented, a write of [TRCIMSPEC<n>](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [40:38]

Reserved, RES0.

TRCCNTVRn, bit [37]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented), TRCCNTVR<n> are implemented and System register access to the PE Trace Unit registers is implemented:

Trap MSR writes of [TRCCNTVR<n>](#) at EL1 using AArch64 to EL2.

| TRCCNTVRn | Meaning |
|-----------|--|
| 0b0 | MSR writes of TRCCNTVR<n> are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCCNTVR<n> at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

If Counter n is not implemented, a write of [TRCCNTVR<n>](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCCLAIM, bit [36]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [TRCCLAIMCLR](#).
- [TRCCLAIMSET](#).

| TRCCLAIM | Meaning |
|----------|---|
| 0b0 | MSR writes of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TRCAUXCTLR, bit [35]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MSR writes of [TRCAUXCTLR](#) at EL1 using AArch64 to EL2.

| TRCAUXCTLR | Meaning |
|------------|---|
| 0b0 | MSR writes of TRCAUXCTLR are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TRCAUXCTLR at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [34]

Reserved, RES0.

TRC, bit [33]

When (FEAT_ETE is implemented or FEAT_ETMv4 is implemented) and System register access to the PE Trace Unit registers is implemented:

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [TRCACATR<n>](#).
- [TRCACVR<n>](#).
- [TRCBBCTLR](#).
- [TRCCCCTLR](#).
- [TRCCIDCCTLR0](#).
- [TRCCIDCCTLR1](#).
- [TRCCIDCVR<n>](#).
- [TRCCNTCTLR<n>](#).
- [TRCCNTRLDVR<n>](#).
- [TRCCONFIGR](#).
- [TRCEVENTCTL0R](#).
- [TRCEVENTCTL1R](#).
- TRCEXTINSELR, if FEAT_ETMv4 is implemented.
- [TRCEXTINSELR<n>](#), if FEAT_ETE is implemented.
- [TRCQCTLR](#).
- [TRCRSCTLR<n>](#).
- [TRCRSR](#), if FEAT_ETE is implemented.
- [TRCSEQEVR<n>](#).
- [TRCSEQRSTEV](#).
- [TRCSSCCR<n>](#).
- [TRCSSPCICR<n>](#).
- [TRCSTALLCTLR](#).
- [TRCSYNCPR](#).
- [TRCTRACEIDR](#).
- [TRCTSCTLR](#).
- [TRCVIIECTLR](#).
- [TRCVIPCSSCTLR](#).
- [TRCVISSCTLR](#).
- [TRCVMIDCCTLR0](#).
- [TRCVMIDCCTLR1](#).
- [TRCVMIDCVR<n>](#).

| TRC | Meaning |
|-----|---|
| 0b0 | MSR writes of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

A write of an unimplemented register is UNDEFINED.

[TRCEXTINSELR<n>](#) and [TRCRSR](#) are only implemented if FEAT_ETE is implemented.

TRCEXTINSELR is only implemented if FEAT_ETE is not implemented and FEAT_ETMv4 is implemented.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSLATFR_EL1, bit [32]

When FEAT_SPE is implemented:

Trap MSR writes of [PMSLATFR_EL1](#) at EL1 using AArch64 to EL2.

| PMSLATFR_EL1 | Meaning |
|--------------|---|
| 0b0 | MSR writes of PMSLATFR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMSLATFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSIRR_EL1, bit [31]

When FEAT_SPE is implemented:

Trap MSR writes of [PMSIRR_EL1](#) at EL1 using AArch64 to EL2.

| PMSIRR_EL1 | Meaning |
|------------|---|
| 0b0 | MSR writes of PMSIRR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMSIRR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [30]

Reserved, RES0.

PMSICR_EL1, bit [29]

When FEAT_SPE is implemented:

Trap MSR writes of [PMSICR_EL1](#) at EL1 using AArch64 to EL2.

| PMSICR_EL1 | Meaning |
|------------|---|
| 0b0 | MSR writes of PMSICR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMSICR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSFCR_EL1, bit [28]

When FEAT_SPE is implemented:

Trap MSR writes of [PMSFCR_EL1](#) at EL1 using AArch64 to EL2.

| PMSFCR_EL1 | Meaning |
|------------|---|
| 0b0 | MSR writes of PMSFCR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMSFCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSEVFR_EL1, bit [27]**When FEAT_SPE is implemented:**

Trap MSR writes of [PMSEVFR_EL1](#) at EL1 using AArch64 to EL2.

| PMSEVFR_EL1 | Meaning |
|--------------------|--|
| 0b0 | MSR writes of PMSEVFR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMSEVFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSCR_EL1, bit [26]**When FEAT_SPE is implemented:**

Trap MSR writes of [PMSCR_EL1](#) at EL1 using AArch64 to EL2.

| PMSCR_EL1 | Meaning |
|------------------|--|
| 0b0 | MSR writes of PMSCR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMSCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMBSR_EL1, bit [25]**When FEAT_SPE is implemented:**

Trap MSR writes of [PMBSR_EL1](#) at EL1 using AArch64 to EL2.

| PMBSR_EL1 | Meaning |
|------------------|--|
| 0b0 | MSR writes of PMBSR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMBSR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMBPTR_EL1, bit [24]

When FEAT_SPE is implemented:

Trap MSR writes of [PMBPTR_EL1](#) at EL1 using AArch64 to EL2.

| PMBPTR_EL1 | Meaning |
|-------------------|---|
| 0b0 | MSR writes of PMBPTR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMBPTR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMBLIMITR_EL1, bit [23]

When FEAT_SPE is implemented:

Trap MSR writes of [PMBLIMITR_EL1](#) at EL1 using AArch64 to EL2.

| PMBLIMITR_EL1 | Meaning |
|----------------------|--|
| 0b0 | MSR writes of PMBLIMITR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PMBLIMITR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [22]

Reserved, RES0.

PMCR_EL0, bit [21]**When FEAT_PMUv3 is implemented:**

Trap MSR writes of [PMCR_EL0](#) at EL1 and EL0 using AArch64 and MCR writes of [PMCR](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| PMCR_EL0 | Meaning |
|----------|---|
| 0b0 | MSR writes of PMCR_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMCR at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTE == 1, then, unless the write generates a higher priority exception: <ul style="list-style-type: none"> MSR writes of PMCR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MCR writes of PMCR at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSWINC_EL0, bit [20]**When FEAT_PMUv3 is implemented:**

Trap MSR writes of [PMSWINC_EL0](#) at EL1 and EL0 using AArch64 and MCR writes of [PMSWINC](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| PMSWINC_EL0 | Meaning |
|-------------|---|
| 0b0 | MSR writes of PMSWINC_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMSWINC at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTE == 1, then, unless the write generates a higher priority exception: <ul style="list-style-type: none"> MSR writes of PMSWINC_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MCR writes of PMSWINC at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMSELR_EL0, bit [19]**When FEAT_PMUv3 is implemented:**

Trap MSR writes of [PMSELR_EL0](#) at EL1 and EL0 using AArch64 and MCR writes of [PMSELR](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| PMSELR_EL0 | Meaning |
|------------|---|
| 0b0 | MSR writes of PMSELR_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMSELR at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | <p>If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the write generates a higher priority exception:</p> <ul style="list-style-type: none"> MSR writes of PMSELR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MCR writes of PMSELR at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMOVS, bit [18]**When FEAT_PMUv3 is implemented:**

Trap MSR writes and MCR writes of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MSR writes of [PMOVSLR_EL0](#) and [PMOVSSET_EL0](#).
- At EL0 using AArch32 when EL1 is using AArch64: MCR writes of [PMOVS](#) and [PMOVSSET](#).

| PMOVS | Meaning |
|-------|---|
| 0b0 | The operations listed above are not trapped by this mechanism. |
| 0b1 | <p>If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the write generates a higher priority exception:</p> <ul style="list-style-type: none"> MSR writes at EL1 and EL0 using AArch64 of PMOVSLR_EL0 and PMOVSSET_EL0 are trapped to EL2 and reported with EC syndrome value 0x18. MCR writes at EL0 using AArch32 of PMOVS and PMOVSSET are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMINTEN, bit [17]**When FEAT_PMUv3 is implemented:**

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [PMINTENCLR_EL1](#).
- [PMINTENSET_EL1](#).

| PMINTEN | Meaning |
|---------|---|
| 0b0 | MSR writes of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMCNTEN, bit [16]**When FEAT_PMUv3 is implemented:**

Trap MSR writes and MCR writes of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MSR writes of [PMCNTENCLR_EL0](#) and [PMCNTENSET_EL0](#).
- At EL0 using AArch32 when EL1 is using AArch64: MCR writes of [PMCNTENCLR](#) and [PMCNTENSET](#).

| PMCNTEN | Meaning |
|---------|--|
| 0b0 | The operations listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the write generates a higher priority exception: <ul style="list-style-type: none"> • MSR writes at EL1 and EL0 using AArch64 of PMCNTENCLR_EL0 and PMCNTENSET_EL0 are trapped to EL2 and reported with EC syndrome value 0x18. • MCR writes at EL0 using AArch32 of PMCNTENCLR and PMCNTENSET are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMCCNTR_EL0, bit [15]**When FEAT_PMUv3 is implemented:**

Trap MSR writes of [PMCCNTR_EL0](#) at EL1 and EL0 using AArch64 and MCR and MCRR writes of [PMCCNTR](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| PMCCNTR_EL0 | Meaning |
|--------------------|---|
| 0b0 | MSR writes of PMCCNTR_EL0 at EL1 and EL0 using AArch64 and MCR and MCRR writes of PMCCNTR at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | <p>If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the write generates a higher priority exception:</p> <ul style="list-style-type: none"> MSR writes of PMCCNTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MCR and MCRR writes of PMCCNTR at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03 (for MCR) or 0x04 (for MCRR). |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMCCFILTR_EL0, bit [14]**When FEAT_PMUv3 is implemented:**

Trap MSR writes of [PMCCFILTR_EL0](#) at EL1 and EL0 using AArch64 and MCR writes of [PMCCFILTR](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| PMCCFILTR_EL0 | Meaning |
|----------------------|---|
| 0b0 | MSR writes of PMCCFILTR_EL0 at EL1 and EL0 using AArch64 and MCR writes of PMCCFILTR at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | <p>If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the write generates a higher priority exception:</p> <ul style="list-style-type: none"> MSR writes of PMCCFILTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MCR writes of PMCCFILTR at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

[PMCCFILTR_EL0](#) can also be accessed in AArch64 state using [PMXEVTYPYPER_EL0](#) when [PMSELR_EL0](#).SEL == 31, and [PMCCFILTR](#) can also be accessed in AArch32 state using [PMXEVTYPYPER](#) when [PMSELR](#).SEL == 31.

Setting this field to 1 has no effect on accesses to [PMXEVTYPYPER_EL0](#) and [PMXEVTYPYPER](#), regardless of the value of [PMSELR_EL0](#).SEL or [PMSELR](#).SEL.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMEVTYPEPn_EL0, bit [13]

When FEAT_PMuV3 is implemented:

Trap MSR writes and MCR writes of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MSR writes of [PMEVTYPEP<n>_EL0](#) and [PMXEVTYPEn_EL0](#).
- At EL0 using AArch32 when EL1 is using AArch64: MCR writes of [PMEVTYPEP<n>](#) and [PMXEVTYPEn](#).

| PMEVTYPEPn_EL0 | Meaning |
|-----------------------|--|
| 0b0 | The operations listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then, unless the write generates a higher priority exception: <ul style="list-style-type: none"> • MSR writes at EL1 and EL0 using AArch64 of PMEVTYPEP<n>_EL0 and PMXEVTYPEn_EL0 are trapped to EL2 and reported with EC syndrome value 0x18. • MCR writes at EL0 using AArch32 of PMEVTYPEP<n> and PMXEVTYPEn are trapped to EL2 and reported with EC syndrome value 0x03. |

Regardless of the value of this field, for each value n:

- If event counter n is not implemented, the following accesses are UNDEFINED:
 - In AArch64 state, a write of [PMEVTYPEP<n>_EL0](#), or, if n is not 31, a write of [PMXEVTYPEn_EL0](#) when [PMSELR_EL0](#).SEL == n.
 - In AArch32 state, a write of [PMEVTYPEP<n>](#), or, if n is not 31, a write of [PMXEVTYPEn](#) when [PMSELR](#).SEL == n.
- If event counter n is implemented, n is greater-than-or-equal-to [MDCR_EL2](#).HPMN, and EL2 is implemented and enabled in the current Security state, the following generate a Trap exception to EL2 from EL0 or EL1:
 - In AArch64 state, a write of [PMEVTYPEP<n>_EL0](#), or a write of [PMXEVTYPEn_EL0](#) when [PMSELR_EL0](#).SEL == n, reported with EC syndrome value 0x18.
 - In AArch32 state, a write of [PMEVTYPEP<n>](#), or a write of [PMXEVTYPEn](#) when [PMSELR](#).SEL == n, reported with EC syndrome value 0x03.

See also HDFGWTR_EL2.PMCCFILTR_EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

PMEVCNTRn_EL0, bit [12]

When FEAT_PMuV3 is implemented:

Trap MSR writes and MCR writes of multiple System registers.

Enables a trap to EL2 the following operations:

- At EL1 and EL0 using AArch64: MSR writes of [PMEVCNTR<n>_EL0](#) and [PMXEVCNTRn_EL0](#).

- At EL0 using AArch32 when EL1 is using AArch64: MCR writes of [PMEVCNTR<n>](#) and [PMXEVCNTR](#).

| PMEVCNTRn_EL0 | Meaning |
|----------------------|---|
| 0b0 | The operations listed above are not trapped by this mechanism. |
| 0b1 | <p>If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the write generates a higher priority exception:</p> <ul style="list-style-type: none"> MSR writes at EL1 and EL0 using AArch64 of PMEVCNTR<n>_EL0 and PMXEVCNTR_EL0 are trapped to EL2 and reported with EC syndrome value 0x18. MCR writes at EL0 using AArch32 of PMEVCNTR<n> and PMXEVCNTR are trapped to EL2 and reported with EC syndrome value 0x03. |

Regardless of the value of this field, for each value n:

- If event counter n is not implemented, the following accesses are UNDEFINED:
 - In AArch64 state, a write of [PMEVCNTR<n>_EL0](#), or a write of [PMXEVCNTR_EL0](#) when [PMSELR_EL0](#).SEL == n.
 - In AArch32 state, a write of [PMEVCNTR<n>](#), or a write of [PMXEVCNTR](#) when [PMSELR](#).SEL == n.
- If event counter n is implemented, n is greater-than-or-equal-to [MDCR_EL2](#).HPMN, and EL2 is implemented and enabled in the current Security state, the following generate a Trap exception to EL2 from EL0 or EL1:
 - In AArch64 state, a write of [PMEVCNTR<n>_EL0](#), or a write of [PMXEVCNTR_EL0](#) when [PMSELR_EL0](#).SEL == n, reported with EC syndrome value 0x18.
 - In AArch32 state, a write of [PMEVCNTR<n>](#), or a write of [PMXEVCNTR](#) when [PMSELR](#).SEL == n, reported with EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

OSDLR_EL1, bit [11]

When FEAT_DoubleLock is implemented:

Trap MSR writes of [OSDLR_EL1](#) at EL1 using AArch64 to EL2.

| OSDLR_EL1 | Meaning |
|------------------|---|
| 0b0 | MSR writes of OSDLR_EL1 are not trapped by this mechanism. |
| 0b1 | <p>If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of OSDLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception.</p> |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

OSECCR_EL1, bit [10]

Trap MSR writes of [OSECCR_EL1](#) at EL1 using AArch64 to EL2.

| OSECCR_EL1 | Meaning |
|-------------------|---|
| 0b0 | MSR writes of OSECCR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of OSECCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Bit [9]

Reserved, RES0.

OSLAR_EL1, bit [8]

Trap MSR writes of [OSLAR_EL1](#) at EL1 using AArch64 to EL2.

| OSLAR_EL1 | Meaning |
|------------------|--|
| 0b0 | MSR writes of OSLAR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of OSLAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DBGPRCR_EL1, bit [7]

Trap MSR writes of [DBGPRCR_EL1](#) at EL1 using AArch64 to EL2.

| DBGPRCR_EL1 | Meaning |
|--------------------|--|
| 0b0 | MSR writes of DBGPRCR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of DBGPRCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Bit [6]

Reserved, RES0.

DBGCLAIM, bit [5]

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [DBGCLAIMCLR_EL1](#).
- [DBGCLAIMSET_EL1](#).

| DBGCLAIM | Meaning |
|----------|---|
| 0b0 | MSR writes of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

MDSCR_EL1, bit [4]

Trap MSR writes of [MDSCR_EL1](#) at EL1 using AArch64 to EL2.

| MDSCR_EL1 | Meaning |
|-----------|--|
| 0b0 | MSR writes of MDSCR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of MDSCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DBGWVRn_EL1, bit [3]

Trap MSR writes of [DBGWVR<n>_EL1](#) at EL1 using AArch64 to EL2.

| DBGWVRn_EL1 | Meaning |
|-------------|--|
| 0b0 | MSR writes of DBGWVR<n>_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of DBGWVR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

If watchpoint n is not implemented, a write of [DBGWVR<n>_EL1](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DBGWCRn_EL1, bit [2]

Trap MSR writes of [DBGWCR<n>_EL1](#) at EL1 using AArch64 to EL2.

| DBGWCRn_EL1 | Meaning |
|-------------|--|
| 0b0 | MSR writes of DBGWCR<n>_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of DBGWCR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

If watchpoint n is not implemented, a write of [DBGWCR<n>_EL1](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DBGBVRn_EL1, bit [1]

Trap MSR writes of [DBGBVR<n>_EL1](#) at EL1 using AArch64 to EL2.

| DBGBVRn_EL1 | Meaning |
|-------------|--|
| 0b0 | MSR writes of DBGBVR<n>_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of DBGBVR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

If breakpoint n is not implemented, a write of [DBGBVR<n>_EL1](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DBGBCRn_EL1, bit [0]

Trap MSR writes of [DBGBCR<n>_EL1](#) at EL1 using AArch64 to EL2.

| DBGBCRn_EL1 | Meaning |
|-------------|--|
| 0b0 | MSR writes of DBGBCR<n>_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of DBGBCR<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

If breakpoint n is not implemented, a write of [DBGBCR<n>_EL1](#) is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Accessing HDFGWTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HDFGWTR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0011 | 0b0001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x1D8];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return HDFGWTR_EL2;
elsif PSTATE.EL == EL3 then
    return HDFGWTR_EL2;

```

MSR HDFGWTR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0011 | 0b0001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x1D8] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HDFGWTR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    HDFGWTR_EL2 = X[t];

```

HFGITR_EL2, Hypervisor Fine-Grained Instruction Trap Register

The HFGITR_EL2 characteristics are:

Purpose

Provides instruction trap controls.

Configuration

This register is present only when FEAT_FGT is implemented. Otherwise, direct accesses to HFGITR_EL2 are UNDEFINED.

Attributes

HFGITR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | |
|-------------|--------------|------------|--------------|---------------|---------------|---------------|--------------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 |
| RES0 | | | | | | | nBRBIALL |
| TLBIVAAE1IS | TLBIASIDE1IS | TLBIVAE1IS | TLBIVMALE1IS | TLBIRVAAE1IOS | TLBIRVALE1IOS | TLBIRVAAE1IOS | TLBIRVAE1IOS |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 |

Bits [63:57]

Reserved, RES0.

nBRBIALL, bit [56]

When FEAT_BRBE is implemented:

Trap execution of [BRB IALL](#) at EL1 using AArch64 to EL2.

| nBRBIALL | Meaning |
|----------|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of BRB IALL at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |
| 0b1 | Execution of BRB IALL is not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

nBRBINJ, bit [55]**When FEAT_BRBE is implemented:**

Trap execution of [BRB INJ](#) at EL1 using AArch64 to EL2.

| nBRBINJ | Meaning |
|----------------|--|
| 0b0 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of BRB INJ at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |
| 0b1 | Execution of BRB INJ is not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

DCCVAC, bit [54]

Trap execution of multiple instructions. Enables a trap on execution at EL1 and EL0 using AArch64 of any of the following AArch64 instructions to EL2:

- [DC CVAC](#).
- [DC CGVAC](#), if FEAT_MTE is implemented.
- [DC CGDVAC](#), if FEAT_MTE is implemented.

| DCCVAC | Meaning |
|---------------|---|
| 0b0 | Execution of the instructions listed above is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution at EL1 and EL0 using AArch64 of any of the instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

SVC_EL1, bit [53]

Trap execution of SVC at EL1 using AArch64 to EL2.

| SVC_EL1 | Meaning |
|----------------|--|
| 0b0 | Execution of SVC is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of SVC at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x15, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

SVC_EL0, bit [52]

Trap execution of SVC at EL0 using AArch64 and execution of SVC at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| SVC_EL0 | Meaning |
|----------------|--|
| 0b0 | Execution of SVC at EL0 using AArch64 and execution of SVC at EL0 using AArch32 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then, unless the instruction generates a higher priority exception: <ul style="list-style-type: none"> • Execution of SVC at EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x15. • Execution of SVC at EL0 using AArch32 is trapped to EL2 and reported with EC syndrome value 0x11. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

ERET, bit [51]

Trap execution of multiple instructions. Enables a trap on execution at EL1 using AArch64 of any of the following AArch64 instructions to EL2:

- ERET.
- ERETAA, if FEAT_PAuth is implemented.
- ERETAB, if FEAT_PAuth is implemented.

| ERET | Meaning |
|-------------|--|
| 0b0 | Execution of the instructions listed above is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution at EL1 using AArch64 of any of the instructions listed above is trapped to EL2 and reported with EC syndrome value 0x1A, unless the instruction generates a higher priority exception. |

If EL2 is implemented and enabled in the current Security state, [HCR_EL2](#).API == 0, and this field enables a fine-grained trap on the instruction, then execution at EL1 using AArch64 of ERETAA or ERETAB instructions is trapped to EL2 and reported with EC syndrome value 0x1A with its associated ISS field, as the fine-grained trap has higher priority than the trap enabled by [HCR_EL2](#).API == 0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

CPPRCTX, bit [50]

When FEAT_SPECRES is implemented:

Trap execution of [CPP RCTX](#) at EL1 and EL0 using AArch64 and execution of [CPPRCTX](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| CPPRCTX | Meaning |
|---------|--|
| 0b0 | Execution of CPP RCTX at EL1 and EL0 using AArch64 and execution of CPPRCTX at EL0 using AArch32 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTE == 1, then, unless the instruction generates a higher priority exception: <ul style="list-style-type: none"> Execution of CPP RCTX at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18. Execution of CPPRCTX at EL0 using AArch32 is trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

DVPRCTX, bit [49]

When FEAT_SPECRES is implemented:

Trap execution of [DVP RCTX](#) at EL1 and EL0 using AArch64 and execution of [DVPRCTX](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| DVPRCTX | Meaning |
|---------|--|
| 0b0 | Execution of DVP RCTX at EL1 and EL0 using AArch64 and execution of DVPRCTX at EL0 using AArch32 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTE == 1, then, unless the instruction generates a higher priority exception: <ul style="list-style-type: none"> Execution of DVP RCTX at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18. Execution of DVPRCTX at EL0 using AArch32 is trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

CFPRCTX, bit [48]

When FEAT_SPECRES is implemented:

Trap execution of [CFP RCTX](#) at EL1 and EL0 using AArch64 and execution of [CFPRCTX](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| CFPRCTX | Meaning |
|---------|--|
| 0b0 | Execution of CFP RCTX at EL1 and EL0 using AArch64 and execution of CFPRCTX at EL0 using AArch32 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3 .FGTE == 1, then, unless the instruction generates a higher priority exception: <ul style="list-style-type: none"> Execution of CFP RCTX at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18. Execution of CFPRCTX at EL0 using AArch32 is trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIVAALE1, bit [47]

Trap execution of [TLBI VAALE1](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI VAALE1NXS.

| TLBIVAALE1 | Meaning |
|------------|--|
| 0b0 | Execution of TLBI VAALE1 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTE == 1, then execution of TLBI VAALE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TLBIVALE1, bit [46]

Trap execution of [TLBI VALE1](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI VALE1NXS.

| TLBIVALE1 | Meaning |
|-----------|---|
| 0b0 | Execution of TLBI VALE1 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTE == 1, then execution of TLBI VALE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TLBIVAAE1, bit [45]

Trap execution of [TLBI VAAE1](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI VAAE1NXS.

| TLBIVAAE1 | Meaning |
|-----------|--|
| 0b0 | Execution of TLBI VAAE1 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of TLBI VAAE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TLBIASIDE1, bit [44]

Trap execution of [TLBI ASIDE1](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI ASIDE1NXS.

| TLBIASIDE1 | Meaning |
|------------|---|
| 0b0 | Execution of TLBI ASIDE1 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of TLBI ASIDE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TLBIVAE1, bit [43]

Trap execution of [TLBI VAE1](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI VAE1NXS.

| TLBIVAE1 | Meaning |
|----------|---|
| 0b0 | Execution of TLBI VAE1 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of TLBI VAE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TLBIVMALE1, bit [42]

Trap execution of [TLBI VMALLE1](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI VMALLE1NXS.

| TLBIVMALE1 | Meaning |
|------------|--|
| 0b0 | Execution of TLBI VMALE1 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VMALE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TLBIRVALE1, bit [41]

When FEAT_TLBIRANGE is implemented:

Trap execution of [TLBI RVALE1](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2.FGTnXS](#) == 0, this field also traps execution of TLBI RVALE1NXS.

| TLBIRVALE1 | Meaning |
|------------|--|
| 0b0 | Execution of TLBI RVALE1 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVALE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIRVALE1, bit [40]

When FEAT_TLBIRANGE is implemented:

Trap execution of [TLBI RVALE1](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2.FGTnXS](#) == 0, this field also traps execution of TLBI RVALE1NXS.

| TLBIRVALE1 | Meaning |
|------------|--|
| 0b0 | Execution of TLBI RVALE1 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVALE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIRVAAE1, bit [39]

When FEAT_TLBIRANGE is implemented:

Trap execution of [TLBI RVAAE1](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI RVAAE1NXS.

| TLBIRVAAE1 | Meaning |
|------------|---|
| 0b0 | Execution of TLBI RVAAE1 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of TLBI RVAAE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIRVAE1, bit [38]

When FEAT_TLBIRANGE is implemented:

Trap execution of [TLBI RVAE1](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI RVAE1NXS.

| TLBIRVAE1 | Meaning |
|-----------|--|
| 0b0 | Execution of TLBI RVAE1 is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of TLBI RVAE1 at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIRVAALE1IS, bit [37]

When FEAT_TLBIRANGE is implemented:

Trap execution of [TLBI RVAALE1IS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI RVAALE1ISNXS.

| TLBIRVALE1IS | Meaning |
|--------------|--|
| 0b0 | Execution of TLBI RVALE1IS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVALE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIRVALE1IS, bit [36]

When FEAT_TLBIRANGE is implemented:

Trap execution of [TLBI RVALE1IS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2.FGTnXS](#) == 0, this field also traps execution of TLBI RVALE1ISNXS.

| TLBIRVALE1IS | Meaning |
|--------------|--|
| 0b0 | Execution of TLBI RVALE1IS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVALE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIRVAAE1IS, bit [35]

When FEAT_TLBIRANGE is implemented:

Trap execution of [TLBI RVAAE1IS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2.FGTnXS](#) == 0, this field also traps execution of TLBI RVAAE1ISNXS.

| TLBIRVAAE1IS | Meaning |
|--------------|--|
| 0b0 | Execution of TLBI RVAAE1IS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVAAE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIRVAE1IS, bit [34]

When FEAT_TLBIRANGE is implemented:

Trap execution of [TLBI RVAE1IS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI RVAE1ISNXS.

| TLBIRVAE1IS | Meaning |
|-------------|--|
| 0b0 | Execution of TLBI RVAE1IS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of TLBI RVAE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIVAALE1IS, bit [33]

Trap execution of [TLBI VAALE1IS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI VAALE1ISNXS.

| TLBIVAALE1IS | Meaning |
|--------------|---|
| 0b0 | Execution of TLBI VAALE1IS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of TLBI VAALE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TLBIVALE1IS, bit [32]

Trap execution of [TLBI VALE1IS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI VALE1ISNXS.

| TLBIVALE1IS | Meaning |
|-------------|---|
| 0b0 | Execution of TLBI VALE1IS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VALE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TLBIVAAE1IS, bit [31]

Trap execution of [TLBI VAAE1IS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2.FGTnXS](#) == 0, this field also traps execution of TLBI VAAE1ISNXS.

| TLBIVAAE1IS | Meaning |
|-------------|---|
| 0b0 | Execution of TLBI VAAE1IS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VAAE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TLBIASIDE1IS, bit [30]

Trap execution of [TLBI ASIDE1IS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2.FGTnXS](#) == 0, this field also traps execution of TLBI ASIDE1ISNXS.

| TLBIASIDE1IS | Meaning |
|--------------|--|
| 0b0 | Execution of TLBI ASIDE1IS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI ASIDE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TLBIVAE1IS, bit [29]

Trap execution of [TLBI VAE1IS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2.FGTnXS](#) == 0, this field also traps execution of TLBI VAE1ISNXS.

| TLBIVAE1IS | Meaning |
|------------|--|
| 0b0 | Execution of TLBI VAE1IS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VAE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TLBIVMALE1IS, bit [28]

Trap execution of [TLBI VMALE1IS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2.FGTnXS](#) == 0, this field also traps execution of TLBI VMALE1ISNXS.

| TLBIVMALE1IS | Meaning |
|--------------|--|
| 0b0 | Execution of TLBI VMALE1IS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VMALE1IS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TLBIRVALE1OS, bit [27]

When FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented:

Trap execution of [TLBI RVALE1OS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2.FGTnXS](#) == 0, this field also traps execution of TLBI RVALE1OSNXS.

| TLBIRVALE1OS | Meaning |
|--------------|--|
| 0b0 | Execution of TLBI RVALE1OS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVALE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIRVALE1OS, bit [26]

When FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented:

Trap execution of [TLBI RVALE1OS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2.FGTnXS](#) == 0, this field also traps execution of TLBI RVALE1OSNXS.

| TLBIRVALE1OS | Meaning |
|--------------|--|
| 0b0 | Execution of TLBI RVALE1OS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVALE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIRVAAE1OS, bit [25]

When FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented:

Trap execution of [TLBI RVAAE1OS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2.FGTnXS](#) == 0, this field also traps execution of TLBI RVAAE1OSNXS.

| TLBIRVAAE1OS | Meaning |
|--------------|--|
| 0b0 | Execution of TLBI RVAAE1OS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVAAE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIRVAE1OS, bit [24]

When FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented:

Trap execution of [TLBI RVAE1OS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2.FGTnXS](#) == 0, this field also traps execution of TLBI RVAE1OSNXS.

| TLBIRVAE1OS | Meaning |
|-------------|---|
| 0b0 | Execution of TLBI RVAE1OS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI RVAE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIVAALE1OS, bit [23]

When FEAT_TLBIOS is implemented:

Trap execution of [TLBI VAALE1OS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI VAALE1OSNXS.

| TLBIVAALE1OS | Meaning |
|--------------|---|
| 0b0 | Execution of TLBI VAALE1OS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of TLBI VAALE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIVALE1OS, bit [22]

When FEAT_TLBIOS is implemented:

Trap execution of [TLBI VALE1OS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI VALE1OSNXS.

| TLBIVALE1OS | Meaning |
|-------------|--|
| 0b0 | Execution of TLBI VALE1OS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of TLBI VALE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIVAAE1OS, bit [21]**When FEAT_TLBIOS is implemented:**

Trap execution of [TLBI VAAE1OS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI VAAE1OSNXS.

| TLBIVAAE1OS | Meaning |
|-------------|--|
| 0b0 | Execution of TLBI VAAE1OS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of TLBI VAAE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIASIDE1OS, bit [20]**When FEAT_TLBIOS is implemented:**

Trap execution of [TLBI ASIDE1OS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI ASIDE1OSNXS.

| TLBIASIDE1OS | Meaning |
|--------------|---|
| 0b0 | Execution of TLBI ASIDE1OS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of TLBI ASIDE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIVAE1OS, bit [19]**When FEAT_TLBIOS is implemented:**

Trap execution of [TLBI VAE1OS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2](#).FGTnXS == 0, this field also traps execution of TLBI VAE1OSNXS.

| TLBIVAE1OS | Meaning |
|------------|--|
| 0b0 | Execution of TLBI VAE1OS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VAE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

TLBIVMALLE1OS, bit [18]

When FEAT_TLBIOS is implemented:

Trap execution of [TLBI VMALLE1OS](#) at EL1 using AArch64 to EL2.

If FEAT_XS is implemented and [HCRX_EL2.FGTnXS](#) == 0, this field also traps execution of TLBI VMALLE1OSNXS.

| TLBIVMALLE1OS | Meaning |
|---------------|---|
| 0b0 | Execution of TLBI VMALLE1OS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of TLBI VMALLE1OS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ATS1E1WP, bit [17]

When FEAT_PAN2 is implemented:

Trap execution of [AT S1E1WP](#) at EL1 using AArch64 to EL2.

| ATS1E1WP | Meaning |
|----------|--|
| 0b0 | Execution of AT S1E1WP is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of AT S1E1WP at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ATS1E1RP, bit [16]

When FEAT_PAN2 is implemented:

Trap execution of [AT S1E1RP](#) at EL1 using AArch64 to EL2.

| ATS1E1RP | Meaning |
|----------|--|
| 0b0 | Execution of AT S1E1RP is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of AT S1E1RP at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ATS1E0W, bit [15]

Trap execution of [AT S1E0W](#) at EL1 using AArch64 to EL2.

| ATS1E0W | Meaning |
|---------|---|
| 0b0 | Execution of AT S1E0W is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of AT S1E0W at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

ATS1E0R, bit [14]

Trap execution of [AT S1E0R](#) at EL1 using AArch64 to EL2.

| ATS1E0R | Meaning |
|---------|---|
| 0b0 | Execution of AT S1E0R is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of AT S1E0R at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

ATS1E1W, bit [13]

Trap execution of [AT S1E1W](#) at EL1 using AArch64 to EL2.

| ATS1E1W | Meaning |
|----------------|---|
| 0b0 | Execution of AT S1E1W is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of AT S1E1W at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

ATS1E1R, bit [12]

Trap execution of [AT S1E1R](#) at EL1 using AArch64 to EL2.

| ATS1E1R | Meaning |
|----------------|---|
| 0b0 | Execution of AT S1E1R is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution of AT S1E1R at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DCZVA, bit [11]

Trap execution of multiple instructions. Enables a trap on execution at EL1 and EL0 using AArch64 of any of the following AArch64 instructions to EL2:

- [DC ZVA](#).
- [DC GVA](#), if FEAT_MTE is implemented.
- [DC GZVA](#), if FEAT_MTE is implemented.

Note

Unlike [HCR_EL2.TDZ](#), this field has no effect on [DCZID_EL0.DZP](#).

| DCZVA | Meaning |
|--------------|---|
| 0b0 | Execution of the instructions listed above is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then execution at EL1 and EL0 using AArch64 of any of the instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DCCIVAC, bit [10]

Trap execution of multiple instructions. Enables a trap on execution at EL1 and EL0 using AArch64 of any of the following AArch64 instructions to EL2:

- [DC CIVAC](#).
- [DC CIGVAC](#), if FEAT_MTE is implemented.
- [DC CIGDVAC](#), if FEAT_MTE is implemented.

| DCCIVAC | Meaning |
|---------|---|
| 0b0 | Execution of the instructions listed above is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution at EL1 and EL0 using AArch64 of any of the instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DCCVADP, bit [9]

When FEAT_DPB2 is implemented:

Trap execution of multiple instructions. Enables a trap on execution at EL1 and EL0 using AArch64 of any of the following AArch64 instructions to EL2:

- [DC CVADP](#).
- [DC CGVADP](#), if FEAT_MTE is implemented.
- [DC CGDVADP](#), if FEAT_MTE is implemented.

| DCCVADP | Meaning |
|---------|---|
| 0b0 | Execution of the instructions listed above is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution at EL1 and EL0 using AArch64 of any of the instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

DCCVAP, bit [8]

Trap execution of multiple instructions. Enables a trap on execution at EL1 and EL0 using AArch64 of any of the following AArch64 instructions to EL2:

- [DC CVAP](#).
- [DC CGVAP](#), if FEAT_MTE is implemented.
- [DC CGDVAP](#), if FEAT_MTE is implemented.

| DCCVAP | Meaning |
|--------|---|
| 0b0 | Execution of the instructions listed above is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution at EL1 and EL0 using AArch64 of any of the instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DCCVAU, bit [7]

Trap execution of [DC CVAU](#) at EL1 and EL0 using AArch64 to EL2.

| DCCVAU | Meaning |
|--------|--|
| 0b0 | Execution of DC CVAU is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of DC CVAU at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DCCISW, bit [6]

Trap execution of multiple instructions. Enables a trap on execution at EL1 using AArch64 of any of the following AArch64 instructions to EL2:

- [DC CISW](#).
- [DC CIGSW](#), if FEAT_MTE2 is implemented.
- [DC CIGDSW](#), if FEAT_MTE2 is implemented.

| DCCISW | Meaning |
|--------|--|
| 0b0 | Execution of the instructions listed above is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution at EL1 using AArch64 of any of the instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DCCSW, bit [5]

Trap execution of multiple instructions. Enables a trap on execution at EL1 using AArch64 of any of the following AArch64 instructions to EL2:

- [DC CSW](#).
- [DC CGSW](#), if FEAT_MTE2 is implemented.
- [DC CGDSW](#), if FEAT_MTE2 is implemented.

| DCCSW | Meaning |
|-------|--|
| 0b0 | Execution of the instructions listed above is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution at EL1 using AArch64 of any of the instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DCISW, bit [4]

Trap execution of multiple instructions. Enables a trap on execution at EL1 using AArch64 of any of the following AArch64 instructions to EL2:

- [DC ISW](#).
- [DC IGSW](#), if FEAT_MTE2 is implemented.
- [DC IGDSW](#), if FEAT_MTE2 is implemented.

| DCISW | Meaning |
|-------|--|
| 0b0 | Execution of the instructions listed above is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution at EL1 using AArch64 of any of the instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DCIVAC, bit [3]

Trap execution of multiple instructions. Enables a trap on execution at EL1 using AArch64 of any of the following AArch64 instructions to EL2:

- [DC IVAC](#).
- [DC IGVAC](#), if FEAT_MTE2 is implemented.
- [DC IGDVAC](#), if FEAT_MTE2 is implemented.

| DCIVAC | Meaning |
|--------|--|
| 0b0 | Execution of the instructions listed above is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution at EL1 using AArch64 of any of the instructions listed above is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

ICIVAU, bit [2]

Trap execution of [IC IVAU](#) at EL1 and EL0 using AArch64 to EL2.

| ICIVAU | Meaning |
|--------|--|
| 0b0 | Execution of IC IVAU is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of IC IVAU at EL1 and EL0 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

ICIALLU, bit [1]

Trap execution of [IC IALLU](#) at EL1 using AArch64 to EL2.

| ICIALLU | Meaning |
|---------|--|
| 0b0 | Execution of IC IALLU is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of IC IALLU at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

ICIALUIS, bit [0]

Trap execution of [IC IALUIS](#) at EL1 using AArch64 to EL2.

| ICIALUIS | Meaning |
|----------|---|
| 0b0 | Execution of IC IALUIS is not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then execution of IC IALUIS at EL1 using AArch64 is trapped to EL2 and reported with EC syndrome value 0x18, unless the instruction generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Accessing HFGITR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HFGITR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x1C8];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return HFGITR_EL2;
elsif PSTATE.EL == EL3 then
    return HFGITR_EL2;

```

MSR HFGITR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x1C8] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HFGITR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    HFGITR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HFGTR_EL2, Hypervisor Fine-Grained Read Trap Register

The HFGTR_EL2 characteristics are:

Purpose

Provides controls for traps of MRS and MRC reads of System registers.

Configuration

This register is present only when FEAT_FGT is implemented. Otherwise, direct accesses to HFGTR_EL2 are UNDEFINED.

Attributes

HFGTR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | |
|-------------|-------------|-----------|------------|---------|-----------|----------|----------|-------------|--------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 |
| RES0 | | | | | | | | nTPIDR2_EL0 | nSMPRI |
| SCXTNUM_EL0 | SCXTNUM_EL1 | SCTLR_EL1 | REVIDR_EL1 | PAR_EL1 | MPIDR_EL1 | MIDR_EL1 | MAIR_EL1 | LORSA_EL1 | LORN_E |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 |

Bits [63:56]

Reserved, RES0.

nTPIDR2_EL0, bit [55]

When FEAT_SME is implemented:

Trap MRS reads of [TPIDR2_EL0](#) at EL1 and EL0 using AArch64 to EL2.

| nTPIDR2_EL0 | Meaning |
|-------------|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then MRS reads of TPIDR2_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of TPIDR2_EL0 are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

nSMPRI_EL1, bit [54]**When FEAT_SME is implemented:**

Trap MRS reads of [SMPRI_EL1](#) at EL1 using AArch64 to EL2.

| nSMPRI_EL1 | Meaning |
|-------------------|--|
| 0b0 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of SMPRI_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of SMPRI_EL1 are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [53:51]

Reserved, RES0.

nACCDATA_EL1, bit [50]**When FEAT_LS64 is implemented:**

Trap MRS reads of [ACCDATA_EL1](#) at EL1 using AArch64 to EL2.

| nACCDATA_EL1 | Meaning |
|---------------------|--|
| 0b0 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ACCDATA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |
| 0b1 | MRS reads of ACCDATA_EL1 are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXADDR_EL1, bit [49]**When FEAT_RAS is implemented:**

Trap MRS reads of [ERXADDR_EL1](#) at EL1 using AArch64 to EL2.

| ERXADDR_EL1 | Meaning |
|-------------|--|
| 0b0 | MRS reads of ERXADDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXADDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXPFPGCDN_EL1, bit [48]

When FEAT_RASv1p1 is implemented:

Trap MRS reads of [ERXPFPGCDN_EL1](#) at EL1 using AArch64 to EL2.

| ERXPFPGCDN_EL1 | Meaning |
|----------------|---|
| 0b0 | MRS reads of ERXPFPGCDN_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXPFPGCDN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXPFPGCTL_EL1, bit [47]

When FEAT_RASv1p1 is implemented:

Trap MRS reads of [ERXPFPGCTL_EL1](#) at EL1 using AArch64 to EL2.

| ERXPFPGCTL_EL1 | Meaning |
|----------------|---|
| 0b0 | MRS reads of ERXPFPGCTL_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXPFPGCTL_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXPFGF_EL1, bit [46]

When FEAT_RAS is implemented:

Trap MRS reads of [ERXPFGF_EL1](#) at EL1 using AArch64 to EL2.

| ERXPFGF_EL1 | Meaning |
|-------------|--|
| 0b0 | MRS reads of ERXPFGF_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXPFGF_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXMISCN_EL1, bit [45]

When FEAT_RAS is implemented:

Trap MRS reads of [ERXMISCN_EL1](#) at EL1 using AArch64 to EL2.

| ERXMISCN_EL1 | Meaning |
|--------------|---|
| 0b0 | MRS reads of ERXMISCN_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXMISCN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXSTATUS_EL1, bit [44]

When FEAT_RAS is implemented:

Trap MRS reads of [ERXSTATUS_EL1](#) at EL1 using AArch64 to EL2.

| ERXSTATUS_EL1 | Meaning |
|----------------------|--|
| 0b0 | MRS reads of ERXSTATUS_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXSTATUS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXCTLR_EL1, bit [43]

When FEAT_RAS is implemented:

Trap MRS reads of [ERXCTLR_EL1](#) at EL1 using AArch64 to EL2.

| ERXCTLR_EL1 | Meaning |
|--------------------|--|
| 0b0 | MRS reads of ERXCTLR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXFR_EL1, bit [42]

When FEAT_RAS is implemented:

Trap MRS reads of [ERXFR_EL1](#) at EL1 using AArch64 to EL2.

| ERXFR_EL1 | Meaning |
|------------------|--|
| 0b0 | MRS reads of ERXFR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERXFR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERRSEL_EL1, bit [41]

When FEAT_RAS is implemented:

Trap MRS reads of [ERRSEL_EL1](#) at EL1 using AArch64 to EL2.

| ERRSEL_EL1 | Meaning |
|------------|---|
| 0b0 | MRS reads of ERRSEL_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERRSEL_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERRIDR_EL1, bit [40]

When FEAT_RAS is implemented:

Trap MRS reads of [ERRIDR_EL1](#) at EL1 using AArch64 to EL2.

| ERRIDR_EL1 | Meaning |
|------------|---|
| 0b0 | MRS reads of ERRIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ERRIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ICC_IGRPENn_EL1, bit [39]

When FEAT_GICv3 is implemented:

Trap MRS reads of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 to EL2.

| ICC_IGRPEN _n _EL1 | Meaning |
|------------------------------|--|
| 0b0 | MRS reads of ICC_IGRPEN<n>_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

VBAR_EL1, bit [38]

Trap MRS reads of [VBAR_EL1](#) at EL1 using AArch64 to EL2.

| VBAR_EL1 | Meaning |
|----------|---|
| 0b0 | MRS reads of VBAR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of VBAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TTBR1_EL1, bit [37]

Trap MRS reads of [TTBR1_EL1](#) at EL1 using AArch64 to EL2.

| TTBR1_EL1 | Meaning |
|-----------|--|
| 0b0 | MRS reads of TTBR1_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TTBR0_EL1, bit [36]

Trap MRS reads of [TTBR0_EL1](#) at EL1 using AArch64 to EL2.

| TTBRO_EL1 | Meaning |
|------------------|--|
| 0b0 | MRS reads of TTBRO_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TTBRO_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TPIDR_EL0, bit [35]

Trap MRS reads of [TPIDR_EL0](#) at EL1 and EL0 using AArch64 and MRC reads of [TPIDRURW](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| TPIDR_EL0 | Meaning |
|------------------|--|
| 0b0 | MRS reads of TPIDR_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURW at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> MRS reads of TPIDR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MRC reads of TPIDRURW at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TPIDRRO_EL0, bit [34]

Trap MRS reads of [TPIDRRO_EL0](#) at EL1 and EL0 using AArch64 and MRC reads of [TPIDRURO](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| TPIDRRO_EL0 | Meaning |
|--------------------|--|
| 0b0 | MRS reads of TPIDRRO_EL0 at EL1 and EL0 using AArch64 and MRC reads of TPIDRURO at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the read generates a higher priority exception: <ul style="list-style-type: none"> MRS reads of TPIDRRO_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MRC reads of TPIDRURO at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TPIDR_EL1, bit [33]

Trap MRS reads of [TPIDR_EL1](#) at EL1 using AArch64 to EL2.

| TPIDR_EL1 | Meaning |
|------------------|--|
| 0b0 | MRS reads of TPIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TPIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TCR_EL1, bit [32]

Trap MRS reads of [TCR_EL1](#) at EL1 using AArch64 to EL2.

| TCR_EL1 | Meaning |
|----------------|--|
| 0b0 | MRS reads of TCR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of TCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

SCXTNUM_EL0, bit [31]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap MRS reads of [SCXTNUM_EL0](#) at EL1 and EL0 using AArch64 to EL2.

| SCXTNUM_EL0 | Meaning |
|--------------------|--|
| 0b0 | MRS reads of SCXTNUM_EL0 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of SCXTNUM_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

SCXTNUM_EL1, bit [30]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap MRS reads of [SCXTNUM_EL1](#) at EL1 using AArch64 to EL2.

| SCXTNUM_EL1 | Meaning |
|-------------|--|
| 0b0 | MRS reads of SCXTNUM_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of SCXTNUM_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

SCTLR_EL1, bit [29]

Trap MRS reads of [SCTLR_EL1](#) at EL1 using AArch64 to EL2.

| SCTLR_EL1 | Meaning |
|-----------|--|
| 0b0 | MRS reads of SCTLR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of SCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

REVIDR_EL1, bit [28]

Trap MRS reads of [REVIDR_EL1](#) at EL1 using AArch64 to EL2.

| REVIDR_EL1 | Meaning |
|------------|---|
| 0b0 | MRS reads of REVIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of REVIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

PAR_EL1, bit [27]

Trap MRS reads of [PAR_EL1](#) at EL1 using AArch64 to EL2.

| PAR_EL1 | Meaning |
|----------------|--|
| 0b0 | MRS reads of PAR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

MPIDR_EL1, bit [26]

Trap MRS reads of [MPIDR_EL1](#) at EL1 using AArch64 to EL2.

| MPIDR_EL1 | Meaning |
|------------------|--|
| 0b0 | MRS reads of MPIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of MPIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

MIDR_EL1, bit [25]

Trap MRS reads of [MIDR_EL1](#) at EL1 using AArch64 to EL2.

| MIDR_EL1 | Meaning |
|-----------------|---|
| 0b0 | MRS reads of MIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of MIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

MAIR_EL1, bit [24]

Trap MRS reads of [MAIR_EL1](#) at EL1 using AArch64 to EL2.

| MAIR_EL1 | Meaning |
|-----------------|---|
| 0b0 | MRS reads of MAIR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of MAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

LORSA_EL1, bit [23]**When FEAT_LOR is implemented:**

Trap MRS reads of [LORSA_EL1](#) at EL1 using AArch64 to EL2.

| LORSA_EL1 | Meaning |
|------------------|--|
| 0b0 | MRS reads of LORSA_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LORSA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

LORN_EL1, bit [22]**When FEAT_LOR is implemented:**

Trap MRS reads of [LORN_EL1](#) at EL1 using AArch64 to EL2.

| LORN_EL1 | Meaning |
|-----------------|---|
| 0b0 | MRS reads of LORN_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LORN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

LORID_EL1, bit [21]**When FEAT_LOR is implemented:**

Trap MRS reads of [LORID_EL1](#) at EL1 using AArch64 to EL2.

| LORID_EL1 | Meaning |
|------------------|--|
| 0b0 | MRS reads of LORID_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LORID_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

LOREA_EL1, bit [20]

When FEAT_LOR is implemented:

Trap MRS reads of [LOREA_EL1](#) at EL1 using AArch64 to EL2.

| LOREA_EL1 | Meaning |
|-----------|--|
| 0b0 | MRS reads of LOREA_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LOREA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

LORC_EL1, bit [19]

When FEAT_LOR is implemented:

Trap MRS reads of [LORC_EL1](#) at EL1 using AArch64 to EL2.

| LORC_EL1 | Meaning |
|----------|---|
| 0b0 | MRS reads of LORC_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of LORC_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ISR_EL1, bit [18]

Trap MRS reads of [ISR_EL1](#) at EL1 using AArch64 to EL2.

| ISR_EL1 | Meaning |
|---------|--|
| 0b0 | MRS reads of ISR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ISR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

FAR_EL1, bit [17]

Trap MRS reads of [FAR_EL1](#) at EL1 using AArch64 to EL2.

| FAR_EL1 | Meaning |
|----------------|--|
| 0b0 | MRS reads of FAR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of FAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

ESR_EL1, bit [16]

Trap MRS reads of [ESR_EL1](#) at EL1 using AArch64 to EL2.

| ESR_EL1 | Meaning |
|----------------|--|
| 0b0 | MRS reads of ESR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of ESR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

DCZID_EL0, bit [15]

Trap MRS reads of [DCZID_EL0](#) at EL1 and EL0 using AArch64 to EL2.

| DCZID_EL0 | Meaning |
|------------------|--|
| 0b0 | MRS reads of DCZID_EL0 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of DCZID_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

CTR_EL0, bit [14]

Trap MRS reads of [CTR_EL0](#) at EL1 and EL0 using AArch64 to EL2.

| CTR_EL0 | Meaning |
|---------|--|
| 0b0 | MRS reads of CTR_EL0 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then MRS reads of CTR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

CSSELR_EL1, bit [13]

Trap MRS reads of [CSSELR_EL1](#) at EL1 using AArch64 to EL2.

| CSSELR_EL1 | Meaning |
|------------|--|
| 0b0 | MRS reads of CSSELR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then MRS reads of CSSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

CPACR_EL1, bit [12]

Trap MRS reads of [CPACR_EL1](#) at EL1 using AArch64 to EL2.

| CPACR_EL1 | Meaning |
|-----------|---|
| 0b0 | MRS reads of CPACR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then MRS reads of CPACR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

CONTEXTIDR_EL1, bit [11]

Trap MRS reads of [CONTEXTIDR_EL1](#) at EL1 using AArch64 to EL2.

| CONTEXTIDR_EL1 | Meaning |
|----------------|--|
| 0b0 | MRS reads of CONTEXTIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then MRS reads of CONTEXTIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

CLIDR_EL1, bit [10]

Trap MRS reads of [CLIDR_EL1](#) at EL1 using AArch64 to EL2.

| CLIDR_EL1 | Meaning |
|-----------|--|
| 0b0 | MRS reads of CLIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CLIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

CCSIDR_EL1, bit [9]

Trap MRS reads of [CCSIDR_EL1](#) at EL1 using AArch64 to EL2.

| CCSIDR_EL1 | Meaning |
|------------|---|
| 0b0 | MRS reads of CCSIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of CCSIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

APIBKey, bit [8]

When FEAT_PAuth is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [APIBKeyHi_EL1](#).
- [APIBKeyLo_EL1](#).

| APIBKey | Meaning |
|---------|---|
| 0b0 | MRS reads of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

APIAKey, bit [7]**When FEAT_PAuth is implemented:**

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [APIAKeyHi_EL1](#).
- [APIAKeyLo_EL1](#).

| APIAKey | Meaning |
|---------|---|
| 0b0 | MRS reads of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

APGAKey, bit [6]**When FEAT_PAuth is implemented:**

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [APGAKeyHi_EL1](#).
- [APGAKeyLo_EL1](#).

| APGAKey | Meaning |
|---------|---|
| 0b0 | MRS reads of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

APDBKey, bit [5]**When FEAT_PAuth is implemented:**

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [APDBKeyHi_EL1](#).
- [APDBKeyLo_EL1](#).

| APDBKey | Meaning |
|---------|---|
| 0b0 | MRS reads of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

APDAKey, bit [4]

When FEAT_PAuth is implemented:

Trap MRS reads of multiple System registers. Enables a trap on MRS reads at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [APDAKeyHi_EL1](#).
- [APDAKeyLo_EL1](#).

| APDAKey | Meaning |
|---------|---|
| 0b0 | MRS reads of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

AMAIR_EL1, bit [3]

Trap MRS reads of [AMAIR_EL1](#) at EL1 using AArch64 to EL2.

| AMAIR_EL1 | Meaning |
|-----------|--|
| 0b0 | MRS reads of AMAIR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MRS reads of AMAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

AIDR_EL1, bit [2]

Trap MRS reads of [AIDR_EL1](#) at EL1 using AArch64 to EL2.

| AIDR_EL1 | Meaning |
|-----------------|--|
| 0b0 | MRS reads of AIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then MRS reads of AIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

AFSR1_EL1, bit [1]

Trap MRS reads of [AFSR1_EL1](#) at EL1 using AArch64 to EL2.

| AFSR1_EL1 | Meaning |
|------------------|---|
| 0b0 | MRS reads of AFSR1_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then MRS reads of AFSR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

AFSR0_EL1, bit [0]

Trap MRS reads of [AFSR0_EL1](#) at EL1 using AArch64 to EL2.

| AFSR0_EL1 | Meaning |
|------------------|---|
| 0b0 | MRS reads of AFSR0_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then MRS reads of AFSR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the read generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Accessing HFGTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HFGTR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x1B8];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return HFGTR_EL2;
elsif PSTATE.EL == EL3 then
    return HFGTR_EL2;

```

MSR HFGTR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x1B8] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HFGTR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    HFGTR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HFGWTR_EL2, Hypervisor Fine-Grained Write Trap Register

The HFGWTR_EL2 characteristics are:

Purpose

Provides controls for traps of MSR and MCR writes of System registers.

Configuration

This register is present only when FEAT_FGT is implemented. Otherwise, direct accesses to HFGWTR_EL2 are UNDEFINED.

Attributes

HFGWTR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | |
|-------------|-------------|-----------|------|---------|------|----------|-----------|-------------|------------|-----------|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 |
| RES0 | | | | | | | | nTPIDR2_EL0 | nSMPRI_EL1 | RES0 | |
| SCXTNUM_EL0 | SCXTNUM_EL1 | SCTLR_EL1 | RES0 | PAR_EL1 | RES0 | MAIR_EL1 | LORSA_EL1 | LORN_EL1 | RES0 | LOREA_EL1 | LO |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |

Bits [63:56]

Reserved, RES0.

nTPIDR2_EL0, bit [55] When FEAT_SME is implemented:

Trap MSR writes of [TPIDR2_EL0](#) at EL1 and EL0 using AArch64 to EL2.

| nTPIDR2_EL0 | Meaning |
|-------------|---|
| 0b0 | If EL2 is implemented and enabled in the current Security state, HCR_EL2 .{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3 .FGTEn == 1, then MSR writes of TPIDR2_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |
| 0b1 | MSR writes of TPIDR2_EL0 are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

nSMPRI_EL1, bit [54]**When FEAT_SME is implemented:**

Trap MSR writes of [SMPRI_EL1](#) at EL1 using AArch64 to EL2.

| nSMPRI_EL1 | Meaning |
|-------------------|--|
| 0b0 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of SMPRI_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |
| 0b1 | MSR writes of SMPRI_EL1 are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [53:51]

Reserved, RES0.

nACCDATA_EL1, bit [50]**When FEAT_LS64 is implemented:**

Trap MSR writes of [ACCDATA_EL1](#) at EL1 using AArch64 to EL2.

| nACCDATA_EL1 | Meaning |
|---------------------|--|
| 0b0 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ACCDATA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |
| 0b1 | MSR writes of ACCDATA_EL1 are not trapped by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXADDR_EL1, bit [49]**When FEAT_RAS is implemented:**

Trap MSR writes of [ERXADDR_EL1](#) at EL1 using AArch64 to EL2.

| ERXADDR_EL1 | Meaning |
|-------------|--|
| 0b0 | MSR writes of ERXADDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXADDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXPFPGCDN_EL1, bit [48]

When FEAT_RASv1p1 is implemented:

Trap MSR writes of [ERXPFPGCDN_EL1](#) at EL1 using AArch64 to EL2.

| ERXPFPGCDN_EL1 | Meaning |
|----------------|---|
| 0b0 | MSR writes of ERXPFPGCDN_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXPFPGCDN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXPFPGCTL_EL1, bit [47]

When FEAT_RASv1p1 is implemented:

Trap MSR writes of [ERXPFPGCTL_EL1](#) at EL1 using AArch64 to EL2.

| ERXPFPGCTL_EL1 | Meaning |
|----------------|---|
| 0b0 | MSR writes of ERXPFPGCTL_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXPFPGCTL_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [46]

Reserved, RES0.

ERXMISCn_EL1, bit [45]

When FEAT_RAS is implemented:

Trap MSR writes of ERXMISC<n>_EL1 at EL1 using AArch64 to EL2.

| ERXMISCn_EL1 | Meaning |
|--------------|---|
| 0b0 | MSR writes of ERXMISC<n>_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXMISC<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXSTATUS_EL1, bit [44]

When FEAT_RAS is implemented:

Trap MSR writes of [ERXSTATUS_EL1](#) at EL1 using AArch64 to EL2.

| ERXSTATUS_EL1 | Meaning |
|---------------|--|
| 0b0 | MSR writes of ERXSTATUS_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXSTATUS_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

ERXCTLR_EL1, bit [43]

When FEAT_RAS is implemented:

Trap MSR writes of [ERXCTLR_EL1](#) at EL1 using AArch64 to EL2.

| ERXCTLR_EL1 | Meaning |
|--------------------|--|
| 0b0 | MSR writes of ERXCTLR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERXCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [42]

Reserved, RES0.

ERRSELR_EL1, bit [41]

When [FEAT_RAS](#) is implemented:

Trap MSR writes of [ERRSELR_EL1](#) at EL1 using AArch64 to EL2.

| ERRSELR_EL1 | Meaning |
|--------------------|--|
| 0b0 | MSR writes of ERRSELR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ERRSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [40]

Reserved, RES0.

ICC_IGRPENn_EL1, bit [39]

When [FEAT_GICv3](#) is implemented:

Trap MSR writes of [ICC_IGRPEN<n>_EL1](#) at EL1 using AArch64 to EL2.

| ICC_IGRPEN _n _EL1 | Meaning |
|------------------------------|--|
| 0b0 | MSR writes of ICC_IGRPEN<n>_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ICC_IGRPEN<n>_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

VBAR_EL1, bit [38]

Trap MSR writes of [VBAR_EL1](#) at EL1 using AArch64 to EL2.

| VBAR_EL1 | Meaning |
|----------|---|
| 0b0 | MSR writes of VBAR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of VBAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TTBR1_EL1, bit [37]

Trap MSR writes of [TTBR1_EL1](#) at EL1 using AArch64 to EL2.

| TTBR1_EL1 | Meaning |
|-----------|--|
| 0b0 | MSR writes of TTBR1_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TTBR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TTBR0_EL1, bit [36]

Trap MSR writes of [TTBR0_EL1](#) at EL1 using AArch64 to EL2.

| TTBR0_EL1 | Meaning |
|------------------|--|
| 0b0 | MSR writes of TTBR0_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TTBR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TPIDR_EL0, bit [35]

Trap MSR writes of [TPIDR_EL0](#) at EL1 and EL0 using AArch64 and MCR writes of [TPIDRURW](#) at EL0 using AArch32 when EL1 is using AArch64 to EL2.

| TPIDR_EL0 | Meaning |
|------------------|---|
| 0b0 | MSR writes of TPIDR_EL0 at EL1 and EL0 using AArch64 and MCR writes of TPIDRURW at EL0 using AArch32 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, EL1 is using AArch64, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then, unless the write generates a higher priority exception: <ul style="list-style-type: none"> MSR writes of TPIDR_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18. MCR writes of TPIDRURW at EL0 using AArch32 are trapped to EL2 and reported with EC syndrome value 0x03. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TPIDRRO_EL0, bit [34]

Trap MSR writes of [TPIDRRO_EL0](#) at EL1 using AArch64 to EL2.

| TPIDRRO_EL0 | Meaning |
|--------------------|--|
| 0b0 | MSR writes of TPIDRRO_EL0 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TPIDRRO_EL0 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TPIDR_EL1, bit [33]

Trap MSR writes of [TPIDR_EL1](#) at EL1 using AArch64 to EL2.

| TPIDR_EL1 | Meaning |
|------------------|--|
| 0b0 | MSR writes of TPIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TPIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

TCR_EL1, bit [32]

Trap MSR writes of [TCR_EL1](#) at EL1 using AArch64 to EL2.

| TCR_EL1 | Meaning |
|----------------|--|
| 0b0 | MSR writes of TCR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of TCR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

SCXTNUM_EL0, bit [31]

When [FEAT_CSV2_2](#) is implemented or [FEAT_CSV2_1p2](#) is implemented:

Trap MSR writes of [SCXTNUM_EL0](#) at EL1 and EL0 using AArch64 to EL2.

| SCXTNUM_EL0 | Meaning |
|--------------------|--|
| 0b0 | MSR writes of SCXTNUM_EL0 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, HCR_EL2.{E2H, TGE} != {1, 1}, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of SCXTNUM_EL0 at EL1 and EL0 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

SCXTNUM_EL1, bit [30]

When [FEAT_CSV2_2](#) is implemented or [FEAT_CSV2_1p2](#) is implemented:

Trap MSR writes of [SCXTNUM_EL1](#) at EL1 using AArch64 to EL2.

| SCXTNUM_EL1 | Meaning |
|--------------------|--|
| 0b0 | MSR writes of SCXTNUM_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of SCXTNUM_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

SCTLR_EL1, bit [29]

Trap MSR writes of [SCTLR_EL1](#) at EL1 using AArch64 to EL2.

| SCTLR_EL1 | Meaning |
|------------------|--|
| 0b0 | MSR writes of SCTLR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of SCTLR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Bit [28]

Reserved, RES0.

PAR_EL1, bit [27]

Trap MSR writes of [PAR_EL1](#) at EL1 using AArch64 to EL2.

| PAR_EL1 | Meaning |
|----------------|--|
| 0b0 | MSR writes of PAR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of PAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Bits [26:25]

Reserved, RES0.

MAIR_EL1, bit [24]

Trap MSR writes of [MAIR_EL1](#) at EL1 using AArch64 to EL2.

| MAIR_EL1 | Meaning |
|-----------------|---|
| 0b0 | MSR writes of MAIR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of MAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

LORSA_EL1, bit [23]

When FEAT_LOR is implemented:

Trap MSR writes of [LORSA_EL1](#) at EL1 using AArch64 to EL2.

| LORSA_EL1 | Meaning |
|------------------|--|
| 0b0 | MSR writes of LORSA_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of LORSA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

LORN_EL1, bit [22]

When FEAT_LOR is implemented:

Trap MSR writes of [LORN_EL1](#) at EL1 using AArch64 to EL2.

| LORN_EL1 | Meaning |
|-----------------|---|
| 0b0 | MSR writes of LORN_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of LORN_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [21]

Reserved, RES0.

LOREA_EL1, bit [20]

When FEAT_LOR is implemented:

Trap MSR writes of [LOREA_EL1](#) at EL1 using AArch64 to EL2.

| LOREA_EL1 | Meaning |
|-----------|--|
| 0b0 | MSR writes of LOREA_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of LOREA_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

LORC_EL1, bit [19]

When FEAT_LOR is implemented:

Trap MSR writes of [LORC_EL1](#) at EL1 using AArch64 to EL2.

| LORC_EL1 | Meaning |
|----------|---|
| 0b0 | MSR writes of LORC_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of LORC_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [18]

Reserved, RES0.

FAR_EL1, bit [17]

Trap MSR writes of [FAR_EL1](#) at EL1 using AArch64 to EL2.

| FAR_EL1 | Meaning |
|----------------|--|
| 0b0 | MSR writes of FAR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of FAR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

ESR_EL1, bit [16]

Trap MSR writes of [ESR_EL1](#) at EL1 using AArch64 to EL2.

| ESR_EL1 | Meaning |
|----------------|--|
| 0b0 | MSR writes of ESR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of ESR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Bits [15:14]

Reserved, RES0.

CSSELR_EL1, bit [13]

Trap MSR writes of [CSSELR_EL1](#) at EL1 using AArch64 to EL2.

| CSSELR_EL1 | Meaning |
|-------------------|---|
| 0b0 | MSR writes of CSSELR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of CSSELR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

CPACR_EL1, bit [12]

Trap MSR writes of [CPACR_EL1](#) at EL1 using AArch64 to EL2.

| CPACR_EL1 | Meaning |
|-----------|--|
| 0b0 | MSR writes of CPACR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of CPACR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

CONTEXTIDR_EL1, bit [11]

Trap MSR writes of [CONTEXTIDR_EL1](#) at EL1 using AArch64 to EL2.

| CONTEXTIDR_EL1 | Meaning |
|----------------|---|
| 0b0 | MSR writes of CONTEXTIDR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of CONTEXTIDR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Bits [10:9]

Reserved, RES0.

APIBKey, bit [8]

When FEAT_PAAuth is implemented:

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [APIBKeyHi_EL1](#).
- [APIBKeyLo_EL1](#).

| APIBKey | Meaning |
|---------|---|
| 0b0 | MSR writes of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

APIAKey, bit [7]**When FEAT_PAuth is implemented:**

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [APIAKeyHi_EL1](#).
- [APIAKeyLo_EL1](#).

| APIAKey | Meaning |
|---------|---|
| 0b0 | MSR writes of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

APGAKey, bit [6]**When FEAT_PAuth is implemented:**

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [APGAKeyHi_EL1](#).
- [APGAKeyLo_EL1](#).

| APGAKey | Meaning |
|---------|---|
| 0b0 | MSR writes of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

APDBKey, bit [5]**When FEAT_PAuth is implemented:**

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [APDBKeyHi_EL1](#).
- [APDBKeyLo_EL1](#).

| APDBKey | Meaning |
|---------|---|
| 0b0 | MSR writes of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

APDAKey, bit [4]

When FEAT_PAuth is implemented:

Trap MSR writes of multiple System registers. Enables a trap on MSR writes at EL1 using AArch64 of any of the following AArch64 System registers to EL2:

- [APDAKeyHi_EL1](#).
- [APDAKeyLo_EL1](#).

| APDAKey | Meaning |
|---------|---|
| 0b0 | MSR writes of the System registers listed above are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes at EL1 using AArch64 of any of the System registers listed above are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

AMAIR_EL1, bit [3]

Trap MSR writes of [AMAIR_EL1](#) at EL1 using AArch64 to EL2.

| AMAIR_EL1 | Meaning |
|-----------|--|
| 0b0 | MSR writes of AMAIR_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of AMAIR_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Bit [2]

Reserved, RES0.

AFSR1_EL1, bit [1]

Trap MSR writes of [AFSR1_EL1](#) at EL1 using AArch64 to EL2.

| AFSR1_EL1 | Meaning |
|------------------|--|
| 0b0 | MSR writes of AFSR1_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of AFSR1_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

AFSR0_EL1, bit [0]

Trap MSR writes of [AFSR0_EL1](#) at EL1 using AArch64 to EL2.

| AFSR0_EL1 | Meaning |
|------------------|--|
| 0b0 | MSR writes of AFSR0_EL1 are not trapped by this mechanism. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, and either EL3 is not implemented or SCR_EL3.FGTEn == 1, then MSR writes of AFSR0_EL1 at EL1 using AArch64 are trapped to EL2 and reported with EC syndrome value 0x18, unless the write generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Accessing HFGWTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HFGWTR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x1C0];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return HFGWTR_EL2;
elsif PSTATE.EL == EL3 then
    return HFGWTR_EL2;

```

MSR HFGWTR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x1C0] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FGTEn == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.FGTEn == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        HFGWTR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    HFGWTR_EL2 = X[t];

```

HPFAR_EL2, Hypervisor IPA Fault Address Register

The HPFAR_EL2 characteristics are:

Purpose

Holds the faulting IPA for some aborts on a stage 2 translation taken to EL2.

Configuration

AArch64 System register HPFAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HPFAR\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

The HPFAR_EL2 is written for:

- Translation or Access faults in the second stage of translation.
- An abort in the second stage of translation performed during the translation table walk of a first stage translation, caused by a Translation fault, an Access flag fault, or a Permission fault.
- A stage 2 Address size fault.
- If FEAT_RME is implemented, a Granule Protection Check fault in the second stage of translation.

For all other exceptions taken to EL2, this register is UNKNOWN.

Note

The address held in this register is an address accessed by the instruction fetch or data access that caused the exception that gave rise to the Instruction Abort or Data Abort. It is the lowest address that gave rise to the fault. Where different faults from different addresses arise from the same instruction, such as for an instruction that loads or stores an unaligned address that crosses a page boundary, the architecture does not prioritize between those different faults.

Attributes

HPFAR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|------|----|----|----|----|----|----|----|----|----|------|--|--|--|--|--|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | | | | | |
| NS | | RES0 | | | | | | | | | | | | | | | | | | FIPA | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | FIPA | | | | | | | | | | RES0 | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | |

Execution at EL1 or EL0 makes HPFAR_EL2 become UNKNOWN.

NS, bit [63]

When FEAT_SEL2 is implemented:

Faulting IPA address space.

| NS | Meaning |
|-----|--|
| 0b0 | Faulting IPA is from the Secure IPA space. |
| 0b1 | Faulting IPA is from the Non-secure IPA space. |

For Data Aborts or Instruction Aborts taken to Non-secure EL2:

- This field is RES0.
- The address is from the Non-secure IPA space.

If FEAT_RME is implemented, for Data Aborts or Instruction Aborts taken to Realm EL2:

- This field is RES0.
- The address is from the Realm IPA space.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

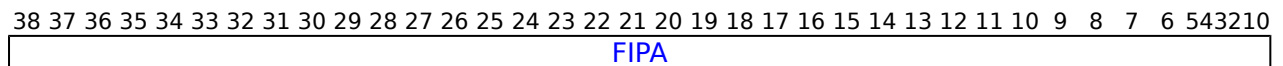
Reserved, RES0.

Bits [62:44]

Reserved, RES0.

FIPA, bits [43:4]

FIPA encoding when FEAT_LPA is implemented



FIPA, bits [38:0]

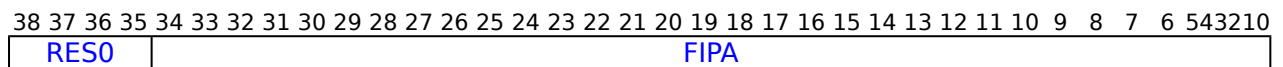
Bits [51:12] of the Faulting Intermediate Physical Address.

For implementations with fewer than 52 physical address bits, the corresponding upper bits in this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FIPA encoding when FEAT_LPA is not implemented



Bits [38:35]

Reserved, RES0.

FIPA, bits [34:0]

Bits[47:12] Faulting Intermediate Physical Address.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [3:0]

Reserved, RES0.

Accessing HPFAR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HPFAR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0110 | 0b0000 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HPFAR_EL2;
elsif PSTATE.EL == EL3 then
    return HPFAR_EL2;

```

MSR HPFAR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0110 | 0b0000 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HPFAR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    HPFAR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HSTR_EL2, Hypervisor System Trap Register

The HSTR_EL2 characteristics are:

Purpose

Controls trapping to EL2 of EL1 or lower AArch32 accesses to the System register in the coproc == 0b1111 encoding space, by the CRn value used to access the register using MCR or MRC instruction. When the register is accessible using an MCRR or MRRC instruction, this is the CRm value used to access the register.

Configuration

AArch64 System register HSTR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HSTR\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

HSTR_EL2 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|------|-----|-----|-----|-----|----|----|----|----|----|------|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | T15 | RES0 | T13 | T12 | T11 | T10 | T9 | T8 | T7 | T6 | T5 | RES0 | T3 | T2 | T1 | T0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16, 14, 4]

Reserved, RES0.

T<n>, bit [n], for n = 15, 13 to 5, 3 to 0

The remaining fields control whether EL0 and EL1 accesses, using MCR, MRC, MCRR, and MRRC instructions, to the System registers in the coproc == 0b1111 encoding space, are trapped to EL2 as follows:

- MCR or MRC accesses to these registers that are trapped to EL2 are reported using EC syndrome value 0x03, unless the access is UNDEFINED.
- MCRR or MRRC accesses to these registers that are trapped to EL2 are reported using EC syndrome value 0x04, unless the access is UNDEFINED.

| T<n> | Meaning |
|------|---|
| 0b0 | This control has no effect on EL0 or EL1 accesses to System registers. |
| 0b1 | <p>System registers in the coproc == 0b1111 encoding space and CRn == <n> or CRm == <n> where T<n> is the name of this field, are trapped as follows:</p> <ul style="list-style-type: none"> • An EL1 MCR or MRC access is trapped to EL2. • An EL0 MCR or MRC access is trapped to EL2, if the access is not UNDEFINED when the value of this field is 0. • An EL1 MCRR or MRRC access is trapped to EL2. • An EL0 MCRR or MRRC access is trapped to EL2, if the access is not UNDEFINED when the value of this field is 0. <p>It is IMPLEMENTATION DEFINED whether an EL0 access using AArch32 is trapped to EL2, or is UNDEFINED.</p> <p>If the access is UNDEFINED, and generates an exception that is taken to EL1 or EL2 using AArch64, this is reported with EC syndrome value 0x00.</p> <hr/> <p>Note</p> <p>Arm expects that trapping to EL2 of EL0 accesses to these registers is unusual and used only when the hypervisor must virtualize EL0 operation. Arm recommends that, whenever possible, EL0 accesses to these registers behave as they would if the implementation did not include EL2. This means that, if the architecture does not support the EL0 access, then the register access instruction is treated as UNDEFINED and generates an exception that is taken to EL1.</p> |

For example, when HSTR_EL2.T7 is 1, for instructions executed at EL1:

- An MCR or MRC instruction with coproc set to 0b1111 and <CRn> set to c7 is trapped to EL2.
- An MCRR or MRRC instruction with coproc set to 0b1111 and <CRm> set to c7 is trapped to EL2.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, RES0.

Accessing HSTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, HSTR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x080];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HSTR_EL2;
elsif PSTATE.EL == EL3 then
    return HSTR_EL2;

```

MSR HSTR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x080] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HSTR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    HSTR_EL2 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

IC IALLU, Instruction Cache Invalidate All to PoU

The IC IALLU characteristics are:

Purpose

Invalidate all instruction caches of the PE executing the instruction to the Point of Unification.

Configuration

AArch64 System instruction IC IALLU performs the same function as AArch32 System instruction [ICIALLU](#).

Attributes

IC IALLU is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the IC IALLU instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

IC IALLU{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TOCU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.ICIALLU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.IC(CacheOpScope_ALLUIS);
    else
        AArch64.IC(CacheOpScope_ALLU);
elsif PSTATE.EL == EL2 then
    AArch64.IC(CacheOpScope_ALLU);
elsif PSTATE.EL == EL3 then
    AArch64.IC(CacheOpScope_ALLU);

```


IC IALLUIS, Instruction Cache Invalidate All to PoU, Inner Shareable

The IC IALLUIS characteristics are:

Purpose

Invalidate all instruction caches in the Inner Shareable domain of the PE executing the instruction to the Point of Unification.

Configuration

AArch64 System instruction IC IALLUIS performs the same function as AArch32 System instruction [ICIALLUIS](#).

Attributes

IC IALLUIS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.
The value in the register specified by <Xt> is ignored.

Executing the IC IALLUIS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

IC IALLUIS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b0111 | 0b0001 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TPU == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TICAB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.ICIALLUIS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.IC(CacheOpScope_ALLUIS);
elsif PSTATE.EL == EL2 then
    AArch64.IC(CacheOpScope_ALLUIS);
elsif PSTATE.EL == EL3 then
    AArch64.IC(CacheOpScope_ALLUIS);
```


IC IVAU, Instruction Cache line Invalidate by VA to PoU

The IC IVAU characteristics are:

Purpose

Invalidate instruction cache by address to Point of Unification.

Configuration

AArch64 System instruction IC IVAU performs the same function as AArch32 System instruction [ICIMVAU](#).

Attributes

IC IVAU is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the IC IVAU instruction

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'The instruction cache maintenance instruction (IC)'.

If EL0 access is enabled, when executed at EL0, if this instruction does not have read access permission to the VA, it is IMPLEMENTATION DEFINED whether it generates a Permission fault.

When FEAT_CMOW is implemented, [HCR_EL2](#).{E2H, TGE} is not {1, 1}, [SCTLR_EL1](#).CMOW is 1, and EL0 is implemented, when executed at EL0, the instruction has stage 1 read permission to the VA, but does not have stage 1 write permission to the VA, the instruction generates a stage 1 Permission fault.

When FEAT_CMOW is implemented, [HCR_EL2](#).E2H is 1, [SCTLR_EL2](#).CMOW is 1, and EL0 access is enabled, when executed at EL0, the instruction has stage 1 read permission to the VA but does not have stage 1 write permission to the VA, the instruction generates a stage 1 Permission fault.

When FEAT_CMOW is implemented, [HCRX_EL2](#).CMOW is 1, and EL1 or EL0 access is enabled, when executed at EL1 or EL0 the instruction has stage 2 read permission to the VA but does not have stage 2 write permission to the VA, the instruction generates a stage 2 Permission fault.

For more information, see 'Permission fault'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

IC IVAU{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b011 | 0b0111 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.UCI == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TPU == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TOCU == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGITR_EL2.ICIVAU == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.UCI == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.IC(X[t], CacheOpScope_PoU);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TPU == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.TOCU == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.ICIVAU == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.IC(X[t], CacheOpScope_PoU);
    elsif PSTATE.EL == EL2 then
        AArch64.IC(X[t], CacheOpScope_PoU);
    elsif PSTATE.EL == EL3 then
        AArch64.IC(X[t], CacheOpScope_PoU);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_AP0R<n>_EL1, Interrupt Controller Active Priorities Group 0 Registers, n = 0 - 3

The ICC_AP0R<n>_EL1 characteristics are:

Purpose

Provides information about Group 0 active priorities.

Configuration

AArch64 System register ICC_AP0R<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ICC_AP0R<n>\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_AP0R<n>_EL1 are UNDEFINED.

Attributes

ICC_AP0R<n>_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

Accessing ICC_AP0R<n>_EL1

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 0 active priorities) might result in UNPREDICTABLE behavior of the interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICC_AP0R1_EL1 is only implemented in implementations that support 6 or more bits of priority. ICC_AP0R2_EL1 and ICC_AP0R3_EL1 are only implemented in implementations that support 7 or more bits of priority. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by [ICH_VTR_EL2.PREbits](#).

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICC_AP0R<n>_EL1.
- Secure [ICC_APIR<n>_EL1](#).
- Non-secure [ICC_APIR<n>_EL1](#).

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_AP0R<n>_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|------------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b1:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        return ICV_AP0R_EL1[UInt(op2<1:0>)];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_AP0R_EL1[UInt(op2<1:0>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_AP0R_EL1[UInt(op2<1:0>)];
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_AP0R_EL1[UInt(op2<1:0>)];

```

MSR ICC_AP0R<n>_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|------------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b1:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        ICV_AP0R_EL1[UInt(op2<1:0>)] = X[t];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_AP1R<n>_EL1, Interrupt Controller Active Priorities Group 1 Registers, n = 0 - 3

The ICC_AP1R<n>_EL1 characteristics are:

Purpose

Provides information about Group 1 active priorities.

Configuration

AArch64 System register ICC_AP1R<n>_EL1 bits [31:0] (S) are architecturally mapped to AArch32 System register [ICC_AP1R<n>\[31:0\]](#) (S).

AArch64 System register ICC_AP1R<n>_EL1 bits [31:0] (NS) are architecturally mapped to AArch32 System register [ICC_AP1R<n>\[31:0\]](#) (NS).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_AP1R<n>_EL1 are UNDEFINED.

Attributes

ICC_AP1R<n>_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| NMI | | RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

NMI, bit [63]

When FEAT_GICv3_NMI is implemented:

Indicates whether there is an active NMI priority.

| NMI | Meaning |
|-----|--|
| 0b0 | There is no active Group 1 NMI, or all active Group 1 NMIs have undergone priority drop. |
| 0b1 | There is an active Group 1 NMI. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [62:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

Accessing ICC_AP1R<n>_EL1

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 1 active priorities) might result in UNPREDICTABLE behavior of the interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICC_AP1R1_EL1 is only implemented in implementations that support 6 or more bits of priority. ICC_AP1R2_EL1 and ICC_AP1R3_EL1 are only implemented in implementations that support 7 or more bits of priority. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by [ICH_VTR_EL2](#).PREbits.

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- [ICC_AP0R<n>_EL1](#).
- Secure ICC_AP1R<n>_EL1.
- Non-secure ICC_AP1R<n>_EL1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_AP1R<n>_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|------------|
| 0b11 | 0b000 | 0b1100 | 0b1001 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        return ICV_AP1R_EL1[UInt(op2<1:0>)];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_AP1R_EL1_S[UInt(op2<1:0>)];
        else
            return ICC_AP1R_EL1_NS[UInt(op2<1:0>)];
    else
        return ICC_AP1R_EL1[UInt(op2<1:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_AP1R_EL1_S[UInt(op2<1:0>)];
        else
            return ICC_AP1R_EL1_NS[UInt(op2<1:0>)];
    else
        return ICC_AP1R_EL1[UInt(op2<1:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            return ICC_AP1R_EL1_S[UInt(op2<1:0>)];
        else
            return ICC_AP1R_EL1_NS[UInt(op2<1:0>)];

```

MSR ICC_AP1R<n>_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|------------|
| 0b11 | 0b000 | 0b1100 | 0b1001 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        ICV_AP1R_EL1[UInt(op2<1:0>)] = X[t];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[UInt(op2<1:0>)] = X[t];
        else
            ICC_AP1R_EL1_NS[UInt(op2<1:0>)] = X[t];
    else
        ICC_AP1R_EL1[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[UInt(op2<1:0>)] = X[t];
        else
            ICC_AP1R_EL1_NS[UInt(op2<1:0>)] = X[t];
    else
        ICC_AP1R_EL1[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[UInt(op2<1:0>)] = X[t];
        else
            ICC_AP1R_EL1_NS[UInt(op2<1:0>)] = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_ASGI1R_EL1, Interrupt Controller Alias Software Generated Interrupt Group 1 Register

The ICC_ASGI1R_EL1 characteristics are:

Purpose

Generates Group 1 SGIs for the Security state that is not the current Security state.

Configuration

AArch64 System register ICC_ASGI1R_EL1 performs the same function as AArch32 System register [ICC_ASGI1R](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_ASGI1R_EL1 are UNDEFINED.

Under certain conditions a write to ICC_ASGI1R_EL1 can generate Group 0 interrupts, see 'Forwarding an SGI to a target PE' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_ASGI1R_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|----|----|------|----|----|----|----|----|----|----|------------|----|----|----|------|----|-----|------|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | Aff3 | | | | | | | | RS | | | | RES0 | | IRM | Aff2 | | | | | | | | |
| RES0 | | | | INTID | | | | Aff1 | | | | | | | | TargetList | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:56]

Reserved, RES0.

Aff3, bits [55:48]

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

RS, bits [47:44]

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value $((RS * 16) + n)$.

When [ICC_CTLR_EL1](#).RSS==0, RS is RES0.

When [ICC_CTLR_EL1](#).RSS==1 and [GICD_TYPER](#).RSS==0, writing this register with $RS \neq 0$ is a CONSTRAINED UNPREDICTABLE choice of :

- The write is ignored.
- The RS field is treated as 0.

Bits [43:41]

Reserved, RES0.

IRM, bit [40]

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

| IRM | Meaning |
|-----|---|
| 0b0 | Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.<target list>. |
| 0b1 | Interrupts routed to all PEs in the system, excluding "self". |

Aff2, bits [39:32]

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]

Reserved, RES0.

INTID, bits [27:24]

The INTID of the SGI.

Aff1, bits [23:16]

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Note

If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

Accessing ICC_ASGI1R_EL1

This register allows software executing in a Secure state to generate Non-secure Group 1 SGIs. It will also allow software executing in a Non-secure state to generate Secure Group 1 SGIs, if permitted by the settings of [GICR_NSACR](#) in the Redistributor corresponding to the target PE.

When [GICD_CTLR](#).DS==0, Non-secure writes do not generate an interrupt for a target PE if not permitted by the [GICR_NSACR](#) register associated with the target PE. For more information, see 'Use of control registers for SGI forwarding'.

Note

Accesses at EL3 are treated as Secure regardless of the value of SCR_EL3.NS.

Accesses to this register use the following encodings in the System register encoding space:

MSR ICC_ASGI1R_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1011 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_ASGI1R_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                ICC_ASGI1R_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_ASGI1R_EL1 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_BPR0_EL1, Interrupt Controller Binary Point Register 0

The ICC_BPR0_EL1 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption.

Configuration

AArch64 System register ICC_BPR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ICC_BPR0\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_BPR0_EL1 are UNDEFINED.

Virtual accesses to this register update [ICH_VMCR_EL2.VBPR0](#).

Attributes

ICC_BPR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. This is done as follows:

| Binary point value | Group priority field | Subpriority field | Field with binary point |
|--------------------|----------------------|-------------------|-------------------------|
| 0 | [7:1] | [0] | ggggggg.s |
| 1 | [7:2] | [1:0] | ggggggg.ss |
| 2 | [7:3] | [2:0] | ggggg.sss |
| 3 | [7:4] | [3:0] | gggg.ssss |
| 4 | [7:5] | [4:0] | ggg.sssss |
| 5 | [7:6] | [5:0] | gg.ssssss |
| 6 | [7] | [6:0] | g.sssssss |
| 7 | No preemption | [7:0] | .ssssssss |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICC_BPR0_EL1

The minimum binary point value is derived from the number of implemented priority bits. The number of priority bits is IMPLEMENTATION DEFINED, and reported by [ICC_CTLR_EL1.PRIBits](#) and [ICC_CTLR_EL3.PRIBits](#).

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_BPR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        return ICV_BPR0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_BPR0_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_BPR0_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_BPR0_EL1;

```

MSR ICC_BPR0_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        ICV_BPR0_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_BPR0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_BPR0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_BPR1_EL1, Interrupt Controller Binary Point Register 1

The ICC_BPR1_EL1 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

Configuration

AArch64 System register ICC_BPR1_EL1 bits [31:0] (S) are architecturally mapped to AArch32 System register [ICC_BPR1\[31:0\]](#) (S).

AArch64 System register ICC_BPR1_EL1 bits [31:0] (NS) are architecturally mapped to AArch32 System register [ICC_BPR1\[31:0\]](#) (NS).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_BPR1_EL1 are UNDEFINED.

Virtual accesses to this register update [ICH_VMCR_EL2.VBPR1](#).

Attributes

ICC_BPR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

If the GIC is configured to use separate binary point fields for Group 0 and Group 1 interrupts, the value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. For more information about priorities, see 'Priority grouping' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The minimum value of the Non-secure copy of this register is the minimum value of [ICC_BPR0_EL1](#) + 1. The minimum value of the Secure copy of this register is the minimum value of [ICC_BPR0_EL1](#).

If EL3 is implemented and [ICC_CTLR_EL3.CBPR_EL1S](#) is 1:

- Accesses to this register from Secure EL2 access the state of [ICC_BPR0_EL1](#).
- Accesses to this register from Secure EL1:
 - When [SCR_EL3.EEL2](#) is 1 and [HCR_EL2.IMO](#) is 1, access the state of [ICV_BPR1_EL1](#).
 - Otherwise, access the state of [ICC_BPR0_EL1](#).

If EL3 is implemented and [ICC_CTLR_EL3](#).CBPR_EL1NS is 1, Non-secure accesses to this register at EL1 or EL2 behave as follows, depending on the values of HCR_EL2.IMO and SCR_EL3.IRQ:

| HCR_EL2.IMO | SCR_EL3.IRQ | Behavior |
|-------------|-------------|--|
| 0b0 | 0b0 | Non-secure EL1 and EL2 reads return ICC_BPR0_EL1 + 1 saturated to 0b111. Non-secure EL1 and EL2 writes are ignored. |
| 0b0 | 0b1 | Non-secure EL1 and EL2 accesses trap to EL3. |
| 0b1 | 0b0 | Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 reads return ICC_BPR0_EL1 + 1 saturated to 0b111. Non-secure EL2 writes are ignored. |
| 0b1 | 0b1 | Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 accesses trap to EL3. |

If EL3 is not implemented and [ICC_CTLR_EL1](#).CBPR is 1, Non-secure accesses to this register at EL1 or EL2 behave as follows, depending on the values of HCR_EL2.IMO:

| HCR_EL2.IMO | Behavior |
|-------------|--|
| 0b0 | Non-secure EL1 and EL2 reads return ICC_BPR0_EL1 + 1 saturated to 0b111. Non-secure EL1 and EL2 writes are ignored. |
| 0b1 | Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 reads return ICC_BPR0_EL1 + 1 saturated to 0b111. Non-secure EL2 writes are ignored. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICC_BPR1_EL1

On a reset, the binary point field is UNKNOWN.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_BPR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        return ICV_BPR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_BPR1_EL1_S;
        else
            return ICC_BPR1_EL1_NS;
    else
        return ICC_BPR1_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_BPR1_EL1_S;
        else
            return ICC_BPR1_EL1_NS;
    else
        return ICC_BPR1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            return ICC_BPR1_EL1_S;
        else
            return ICC_BPR1_EL1_NS;

```

MSR ICC_BPR1_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        ICV_BPR1_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_BPR1_EL1_S = X[t];
        else
            ICC_BPR1_EL1_NS = X[t];
    else
        ICC_BPR1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_BPR1_EL1_S = X[t];
        else
            ICC_BPR1_EL1_NS = X[t];
    else
        ICC_BPR1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_BPR1_EL1_S = X[t];
        else
            ICC_BPR1_EL1_NS = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_CTLR_EL1, Interrupt Controller Control Register (EL1)

The ICC_CTLR_EL1 characteristics are:

Purpose

Controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

Configuration

AArch64 System register ICC_CTLR_EL1 bits [31:0] (S) are architecturally mapped to AArch32 System register [ICC_CTLR\[31:0\]](#) (S).

AArch64 System register ICC_CTLR_EL1 bits [31:0] (NS) are architecturally mapped to AArch32 System register [ICC_CTLR\[31:0\]](#) (NS).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_CTLR_EL1 are UNDEFINED.

Attributes

ICC_CTLR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----------|----|-----|------|-----|------|--------|---------|------|------|------|----|----|----|---------|------|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | ExtRange | | RSS | RES0 | A3V | SEIS | IDbits | PRibits | RES0 | PMHE | RES0 | | | | EOImode | CBPR | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:20]

Reserved, RES0.

ExtRange, bit [19]

Extended INTID range (read-only).

| ExtRange | Meaning |
|---|--|
| 0b0 | CPU interface does not support INTIDs in the range 1024..8191. <ul style="list-style-type: none"> Behaviour is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface. |
| Note Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them. | |
| 0b1 | CPU interface supports INTIDs in the range 1024..8191 <ul style="list-style-type: none"> All INTIDs in the range 1024..8191 are treated as requiring deactivation. |

If EL3 is implemented, ICC_CTLR_EL1.ExtRange is an alias of [ICC_CTLR_EL3.ExtRange](#).

RSS, bit [18]

Range Selector Support. Possible values are:

| RSS | Meaning |
|------------|--|
| 0b0 | Targeted SGIs with affinity level 0 values of 0 - 15 are supported. |
| 0b1 | Targeted SGIs with affinity level 0 values of 0 - 255 are supported. |

This bit is read-only.

Bits [17:16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored. Possible values are:

| A3V | Meaning |
|------------|---|
| 0b0 | The CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers. |
| 0b1 | The CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers. |

If EL3 is implemented, this bit is an alias of [ICC_CTLR_EL3.A3V](#).

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the CPU interface supports local generation of SEIs:

| SEIS | Meaning |
|-------------|--|
| 0b0 | The CPU interface logic does not support local generation of SEIs. |
| 0b1 | The CPU interface logic supports local generation of SEIs. |

If EL3 is implemented, this bit is an alias of [ICC_CTLR_EL3.SEIS](#).

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. The number of physical interrupt identifier bits supported:

| IDbits | Meaning |
|---------------|----------------|
| 0b000 | 16 bits. |
| 0b001 | 24 bits. |

All other values are reserved.

If EL3 is implemented, this field is an alias of [ICC_CTLR_EL3.IDbits](#).

PRIbits, bits [10:8]

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation that supports two Security states must implement at least 32 levels of physical priority (5 priority bits).

An implementation that supports only a single Security state must implement at least 16 levels of physical priority (4 priority bits).

Note

This field always returns the number of priority bits implemented, regardless of the Security state of the access or the value of [GICD_CTLR.DS](#).

For physical accesses, this field determines the minimum value of [ICC_BPR0_EL1](#).

If EL3 is implemented, physical accesses return the value from [ICC_CTLR_EL3.PRIBits](#).

If EL3 is not implemented, physical accesses return the value from this field.

Bit [7]

Reserved, RES0.

PMHE, bit [6]

Priority Mask Hint Enable. Controls whether the priority mask register is used as a hint for interrupt distribution:

| PMHE | Meaning |
|------|---|
| 0b0 | Disables use of ICC_PMR_EL1 as a hint for interrupt distribution. |
| 0b1 | Enables use of ICC_PMR_EL1 as a hint for interrupt distribution. |

If EL3 is implemented, this bit is an alias of [ICC_CTLR_EL3.PMHE](#). Whether this bit can be written as part of an access to this register depends on the value of [GICD_CTLR.DS](#):

- If [GICD_CTLR.DS](#) == 0, this bit is read-only.
- If [GICD_CTLR.DS](#) == 1, this bit is read/write.

If EL3 is not implemented, it is IMPLEMENTATION DEFINED whether this bit is read-only or read-write:

- If this bit is read-only, an implementation can choose to make this field RAZ/WI or RAO/WI.
- If this bit is read/write, it resets to zero.

Bits [5:2]

Reserved, RES0.

EOImode, bit [1]

EOI mode for the current Security state. Controls whether a write to an End of Interrupt register also deactivates the interrupt:

| EOImode | Meaning |
|---------|---|
| 0b0 | ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR_EL1 are UNPREDICTABLE. |
| 0b1 | ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide priority drop functionality only. ICC_DIR_EL1 provides interrupt deactivation functionality. |

The Secure [ICC_CTLR_EL1.EOImode](#) is an alias of [ICC_CTLR_EL3.EOImode_EL1S](#).

The Non-secure [ICC_CTLR_EL1.EOImode](#) is an alias of [ICC_CTLR_EL3.EOImode_EL1NS](#).

CBPR, bit [0]

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 interrupts:

| CBPR | Meaning |
|------|---|
| 0b0 | ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts only. ICC_BPR1_EL1 determines the preemption group for Group 1 interrupts. |
| 0b1 | ICC_BPR0_EL1 determines the preemption group for both Group 0 and Group 1 interrupts. |

If EL3 is implemented:

- This bit is an alias of [ICC_CTLR_EL3](#).CBPR_EL1{S,NS} where S or NS corresponds to the current Security state.
- If [GICD_CTLR](#).DS == 0, this bit is read-only.
- If [GICD_CTLR](#).DS == 1, this bit is read/write.

If EL3 is not implemented, this bit is read/write.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICC_CTLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_CTLR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        return ICV_CTLR_EL1;
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        return ICV_CTLR_EL1;
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_CTLR_EL1_S;
        else
            return ICC_CTLR_EL1_NS;
    else
        return ICC_CTLR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_CTLR_EL1_S;
        else
            return ICC_CTLR_EL1_NS;
    else
        return ICC_CTLR_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            return ICC_CTLR_EL1_S;
        else
            return ICC_CTLR_EL1_NS;

```

MSR ICC_CTLR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        ICV_CTLR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        ICV_CTLR_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t];
        else
            ICC_CTLR_EL1_NS = X[t];
    else
        ICC_CTLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t];
        else
            ICC_CTLR_EL1_NS = X[t];
    else
        ICC_CTLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t];
        else
            ICC_CTLR_EL1_NS = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_CTLR_EL3, Interrupt Controller Control Register (EL3)

The ICC_CTLR_EL3 characteristics are:

Purpose

Controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

Configuration

AArch64 System register ICC_CTLR_EL3 bits [31:0] can be mapped to AArch32 System register [ICC_MCTLR\[31:0\]](#), but this is not architecturally mandated.

This register is present only when FEAT_GICv3 is implemented and EL3 is implemented. Otherwise, direct accesses to ICC_CTLR_EL3 are UNDEFINED.

Attributes

ICC_CTLR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----------|----|-----|-----|------|-----|------|--------|---------|------|------|----|-------------|----|-------------|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | ExtRange | | RSS | nDS | RES0 | A3V | SEIS | IDbits | PRIbits | RES0 | PMHE | RM | EOImode_EL1 | NS | EOImode_EL1 | NS | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 |

Bits [63:20]

Reserved, RES0.

ExtRange, bit [19]

Extended INTID range (read-only).

| ExtRange | Meaning |
|----------|---|
| 0b0 | <p>CPU interface does not support INTIDs in the range 1024..8191.</p> <ul style="list-style-type: none"> Behaviour is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface. <p>Note Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.</p> |
| 0b1 | <p>CPU interface supports INTIDs in the range 1024..8191</p> <ul style="list-style-type: none"> All INTIDs in the range 1024..8191 are treated as requiring deactivation. |

RSS, bit [18]

Range Selector Support.

| RSS | Meaning |
|------------|--|
| 0b0 | Targeted SGIs with affinity level 0 values of 0-15 are supported. |
| 0b1 | Targeted SGIs with affinity level 0 values of 0-255 are supported. |

This bit is read-only.

nDS, bit [17]

Disable Security not supported. Read-only and writes are ignored.

| nDS | Meaning |
|------------|---|
| 0b0 | The CPU interface logic supports disabling of security. |
| 0b1 | The CPU interface logic does not support disabling of security, and requires that security is not disabled. |

When a PE implements the Realm Management Extension, this field is RAO/WI.

Bit [16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored.

| A3V | Meaning |
|------------|--|
| 0b0 | The CPU interface logic does not support non-zero values of the Aff3 field in SGI generation System registers. |
| 0b1 | The CPU interface logic supports non-zero values of the Aff3 field in SGI generation System registers. |

If EL3 is present, [ICC_CTLR_EL1](#).A3V is an alias of ICC_CTLR_EL3.A3V

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the CPU interface supports generation of SEIs:

| SEIS | Meaning |
|-------------|--|
| 0b0 | The CPU interface logic does not support generation of SEIs. |
| 0b1 | The CPU interface logic supports generation of SEIs. |

If EL3 is present, [ICC_CTLR_EL1](#).SEIS is an alias of ICC_CTLR_EL3.SEIS

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. Indicates the number of physical interrupt identifier bits supported.

| IDbits | Meaning |
|---------------|----------------|
| 0b000 | 16 bits. |
| 0b001 | 24 bits. |

All other values are reserved.

If EL3 is present, [ICC_CTLR_EL1](#).IDbits is an alias of ICC_CTLR_EL3.IDbits

PRibits, bits [10:8]

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation that supports two Security states must implement at least 32 levels of physical priority (5 priority bits).

An implementation that supports only a single Security state must implement at least 16 levels of physical priority (4 priority bits).

Note

This field always returns the number of priority bits implemented, regardless of the value of SCR_EL3.NS or the value of [GICD_CTLR.DS](#).

The division between group priority and subpriority is defined in the binary point registers [ICC_BPR0_EL1](#) and [ICC_BPR1_EL1](#).

This field determines the minimum value of ICC_BPR0_EL1.

Bit [7]

Reserved, RES0.

PMHE, bit [6]

Priority Mask Hint Enable.

| PMHE | Meaning |
|------|--|
| 0b0 | Disables use of the priority mask register as a hint for interrupt distribution. |
| 0b1 | Enables use of the priority mask register as a hint for interrupt distribution. |

Software must write [ICC_PMR_EL1](#) to 0xFF before clearing this field to 0.

- An implementation might choose to make this field RAO/WI if priority-based routing is always used
- An implementation might choose to make this field RAZ/WI if priority-based routing is never used

If EL3 is present, [ICC_CTLR_EL1](#).PMHE is an alias of ICC_CTLR_EL3.PMHE.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

RM, bit [5]

Routing Modifier. This bit controls whether EL3 can acknowledge, or observe as the Highest Priority Pending Interrupt, Secure Group 0 and Non-secure Group 1 interrupts.

| RM | Meaning |
|-----|--|
| 0b0 | Secure Group 0 and Non-secure Group 1 interrupts can be acknowledged and observed as the highest priority interrupt at EL3. |
| 0b1 | Secure Group 0 and Non-secure Group 1 interrupts cannot be acknowledged and observed as the highest priority interrupt at EL3. Secure Group 0 interrupts return a special INTID value of 1020. This affects accesses to ICC_IAR0_EL1 and ICC_HPPIR0_EL1 . Non-secure Group 1 interrupts return a special INTID value of 1021. This affects accesses to ICC_IAR1_EL1 and ICC_HPPIR1_EL1 . |

Note

The Routing Modifier bit is supported in AArch64 only. In systems without EL3 the behavior is as if the value is 0. Software must ensure this bit is 0 when the Secure copy of [ICC_SRE_EL1](#).SRE is 1, otherwise system behavior is UNPREDICTABLE. In systems without EL3 or where the Secure copy of [ICC_SRE_EL1](#).SRE is RAO/WI, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EOImode_EL1NS, bit [4]

EOI mode for interrupts handled at Non-secure EL1 and EL2. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

| EOImode_EL1NS | Meaning |
|---------------|---|
| 0b0 | ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR_EL1 are UNPREDICTABLE. |
| 0b1 | ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide priority drop functionality only. ICC_DIR_EL1 provides interrupt deactivation functionality. |

If EL3 is present, [ICC_CTLR_EL1](#)(NS).EOImode is an alias of ICC_CTLR_EL3.EOImode_EL1NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EOImode_EL1S, bit [3]

EOI mode for interrupts handled at Secure EL1 and EL2. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

| EOImode_EL1S | Meaning |
|--------------|---|
| 0b0 | ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR_EL1 are UNPREDICTABLE. |
| 0b1 | ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide priority drop functionality only. ICC_DIR_EL1 provides interrupt deactivation functionality. |

If EL3 is present, [ICC_CTLR_EL1](#)(S).EOImode is an alias of ICC_CTLR_EL3.EOImode_EL1S.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EOImode_EL3, bit [2]

EOI mode for interrupts handled at EL3. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

| EOImode_EL3 | Meaning |
|-------------|---|
| 0b0 | ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR_EL1 are UNPREDICTABLE. |
| 0b1 | ICC_EOIR0_EL1 and ICC_EOIR1_EL1 provide priority drop functionality only. ICC_DIR_EL1 provides interrupt deactivation functionality. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CBPR_EL1NS, bit [1]

Common Binary Point Register, EL1 Non-secure. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 Non-secure interrupts at EL1 and EL2.

| CBPR_EL1NS | Meaning |
|------------|--|
| 0b0 | ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts only. ICC_BPR1_EL1 determines the preemption group for Non-secure Group 1 interrupts. |
| 0b1 | ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts and Non-secure Group 1 interrupts. Non-secure accesses to GICC_BPR and ICC_BPR1_EL1 access the state of ICC_BPR0_EL1 . |

If EL3 is present, [ICC_CTLR_EL1](#)(NS).CBPR is an alias of ICC_CTLR_EL3.CBPR_EL1NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CBPR_EL1S, bit [0]

Common Binary Point Register, EL1 Secure. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 Secure interrupts at EL1 and EL2.

| CBPR_EL1S | Meaning |
|-----------|---|
| 0b0 | ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts only. ICC_BPR1_EL1 determines the preemption group for Secure Group 1 interrupts. |
| 0b1 | ICC_BPR0_EL1 determines the preemption group for Group 0 interrupts and Secure Group 1 interrupts. Secure EL1 accesses to ICC_BPR1_EL1 access the state of ICC_BPR0_EL1 . |

If EL3 is present, [ICC_CTLR_EL1](#)(S).CBPR is an alias of ICC_CTLR_EL3.CBPR_EL1S.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICC_CTLR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_CTLR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1100 | 0b1100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_CTLR_EL3;
    end if
end if

```

MSR ICC_CTLR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1100 | 0b1100 | 0b100 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_CTLR_EL3 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_DIR_EL1, Interrupt Controller Deactivate Interrupt Register

The ICC_DIR_EL1 characteristics are:

Purpose

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified interrupt.

Configuration

AArch64 System register ICC_DIR_EL1 performs the same function as AArch32 System register [ICC_DIR](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_DIR_EL1 are UNDEFINED.

Attributes

ICC_DIR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the interrupt to be deactivated.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR_EL1.IDbits](#) and [ICC_CTLR_EL3.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_DIR_EL1

There are two cases when writing to [ICC_DIR_EL1](#) that were UNPREDICTABLE for a corresponding GICv2 write to [GICC_DIR](#):

- When EOImode == 0. GICv3 implementations must ignore such writes. In systems supporting system error generation, an implementation might generate an SEI.
- When EOImode == 1 but no EOI has been issued. The interrupt will be de-activated by the Distributor, however the active priority in the CPU interface for the interrupt will remain set (because no EOI was issued).

Accesses to this register use the following encodings in the System register encoding space:

MSR ICC_DIR_EL1, <Xt>

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1011 | 0b001 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TDIR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_DIR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_DIR_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_DIR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                ICC_DIR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_DIR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_EOIR0_EL1, Interrupt Controller End Of Interrupt Register 0

The ICC_EOIR0_EL1 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified Group 0 interrupt.

Configuration

AArch64 System register ICC_EOIR0_EL1 performs the same function as AArch32 System register [ICC_EOIR0](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_EOIR0_EL1 are UNDEFINED.

Attributes

ICC_EOIR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding [ICC_IAR0_EL1](#) access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR_EL1.IDbits](#) and [ICC_CTLR_EL3.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the EOImode bit for the current Exception level and Security state is 0, a write to this register drops the priority for the interrupt, and also deactivates the interrupt.

If the EOImode bit for the current Exception level and Security state is 1, a write to this register only drops the priority for the interrupt. Software must write to [ICC_DIR_EL1](#) to deactivate the interrupt.

The EOImode bit for the current Exception level and Security state is determined as follows:

- If EL3 is not implemented, the appropriate bit is [ICC_CTLR_EL1.EOImode](#).
- If EL3 is implemented and the software is executing at EL3, the appropriate bit is [ICC_CTLR_EL3.EOImode_EL3](#).
- If EL3 is implemented and the software is not executing at EL3, the bit depends on the current Security state:
 - If the software is executing in Secure state, the bit is [ICC_CTLR_EL3.EOImode_EL1S](#).
 - If the software is executing in Non-secure state, the bit is [ICC_CTLR_EL3.EOImode_EL1NS](#).

Accessing ICC_EOIR0_EL1

A write to this register must correspond to the most recent valid read by this PE from an Interrupt Acknowledge Register, and must correspond to the INTID that was read from [ICC_IAR0_EL1](#), otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

A write of a Special INTID is ignored. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Accesses to this register use the following encodings in the System register encoding space:

MSR ICC_EOIR0_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        ICV_EOIR0_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR0_EL1 = X[t];

```

ICC_EOIR1_EL1, Interrupt Controller End Of Interrupt Register 1

The ICC_EOIR1_EL1 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified Group 1 interrupt.

Configuration

AArch64 System register ICC_EOIR1_EL1 performs the same function as AArch32 System register [ICC_EOIR1](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_EOIR1_EL1 are UNDEFINED.

Attributes

ICC_EOIR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding [ICC_IAR1_EL1](#) access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR_EL1.IDbits](#) and [ICC_CTLR_EL3.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the EOImode bit for the current Exception level and Security state is 0, a write to this register drops the priority for the interrupt, and also deactivates the interrupt.

If the EOImode bit for the current Exception level and Security state is 1, a write to this register only drops the priority for the interrupt. Software must write to [ICC_DIR_EL1](#) to deactivate the interrupt.

The EOImode bit for the current Exception level and Security state is determined as follows:

- If EL3 is not implemented, the appropriate bit is [ICC_CTLR_EL1.EOImode](#).
- If EL3 is implemented and the software is executing at EL3, the appropriate bit is [ICC_CTLR_EL3.EOImode_EL3](#).
- If EL3 is implemented and the software is not executing at EL3, the bit depends on the current Security state:
 - If the software is executing in Secure state, the bit is [ICC_CTLR_EL3.EOImode_EL1S](#).
 - If the software is executing in Non-secure state, the bit is [ICC_CTLR_EL3.EOImode_EL1NS](#).

Accessing ICC_EOIR1_EL1

A write to this register must correspond to the most recent valid read by this PE from an Interrupt Acknowledge Register, and must correspond to the INTID that was read from [ICC_IAR1_EL1](#), otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

A write of a Special INTID is ignored. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Accesses to this register use the following encodings in the System register encoding space:

MSR ICC_EOIR1_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_EOIR1_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t];

```

ICC_HPPIR0_EL1, Interrupt Controller Highest Priority Pending Interrupt Register 0

The ICC_HPPIR0_EL1 characteristics are:

Purpose

Indicates the highest priority pending Group 0 interrupt on the CPU interface.

Configuration

AArch64 System register ICC_HPPIR0_EL1 performs the same function as AArch32 System register [ICC_HPPIR0](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_HPPIR0_EL1 are UNDEFINED.

Attributes

ICC_HPPIR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending interrupt, if that interrupt is observable at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR_EL1.IDbits](#) and [ICC_CTLR_EL3.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_HPPIR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_HPPIR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        return ICV_HPPIR0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR0_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_HPPIR0_EL1;
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR0_EL1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_HPPIR1_EL1, Interrupt Controller Highest Priority Pending Interrupt Register 1

The ICC_HPPIR1_EL1 characteristics are:

Purpose

Indicates the highest priority pending Group 1 interrupt on the CPU interface.

Configuration

AArch64 System register ICC_HPPIR1_EL1 performs the same function as AArch32 System register [ICC_HPPIR1](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_HPPIR1_EL1 are UNDEFINED.

Attributes

ICC_HPPIR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending interrupt, if that interrupt is observable at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR_EL1.IDbits](#) and [ICC_CTLR_EL3.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_HPPIR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_HPPIR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        return ICV_HPPIR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR1_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_HPPIR1_EL1;
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR1_EL1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_IAR0_EL1, Interrupt Controller Interrupt Acknowledge Register 0

The ICC_IAR0_EL1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled Group 0 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch64 System register ICC_IAR0_EL1 performs the same function as AArch32 System register [ICC_IAR0](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_IAR0_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when $PSTATE.\{I,F\} == \{0,0\}$). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers'.

Attributes

ICC_IAR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

This is the INTID of the highest priority pending interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR_EL1.IDbits](#) and [ICC_CTLR_EL3.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_IAR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_IAR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        return ICV_IAR0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IAR0_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IAR0_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IAR0_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_IAR1_EL1, Interrupt Controller Interrupt Acknowledge Register 1

The ICC_IAR1_EL1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled Group 1 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch64 System register ICC_IAR1_EL1 performs the same function as AArch32 System register [ICC_IAR1](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_IAR1_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when $PSTATE.\{I,F\} == \{0,0\}$). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_IAR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

This is the INTID of the highest priority pending interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR_EL1.IDbits](#) and [ICC_CTLR_EL3.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_IAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_IAR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        return ICV_IAR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IAR1_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IAR1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IAR1_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_IGRPEN0_EL1, Interrupt Controller Interrupt Group 0 Enable register

The ICC_IGRPEN0_EL1 characteristics are:

Purpose

Controls whether Group 0 interrupts are enabled or not.

Configuration

AArch64 System register ICC_IGRPEN0_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ICC_IGRPEN0\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_IGRPEN0_EL1 are UNDEFINED.

Attributes

ICC_IGRPEN0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Enable |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:1]

Reserved, RES0.

Enable, bit [0]

Enables Group 0 interrupts.

| Enable | Meaning |
|--------|----------------------------------|
| 0b0 | Group 0 interrupts are disabled. |
| 0b1 | Group 0 interrupts are enabled. |

Virtual accesses to this register update [ICH_VMCR_EL2.VENG0](#).

If the highest priority pending interrupt for that PE is a Group 0 interrupt using 1 of N model, then the interrupt will be targeted to another PE as a result of the Enable bit changing from 1 to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_IGRPEN0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_IGRPEN0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ICC_IGRPENn_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        return ICV_IGRPEN0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_IGRPEN0_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_IGRPEN0_EL1;
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_IGRPEN0_EL1;

```

MSR ICC_IGRPEN0_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ICC_IGRPENn_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        ICV_IGRPEN0_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_IGRPEN0_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                ICC_IGRPEN0_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_IGRPEN0_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_IGRPEN1_EL1, Interrupt Controller Interrupt Group 1 Enable register

The ICC_IGRPEN1_EL1 characteristics are:

Purpose

Controls whether Group 1 interrupts are enabled for the current Security state.

Configuration

AArch64 System register ICC_IGRPEN1_EL1 bits [31:0] (S) are architecturally mapped to AArch32 System register [ICC_IGRPEN1\[31:0\]](#) (S).

AArch64 System register ICC_IGRPEN1_EL1 bits [31:0] (NS) are architecturally mapped to AArch32 System register [ICC_IGRPEN1\[31:0\]](#) (NS).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_IGRPEN1_EL1 are UNDEFINED.

Attributes

ICC_IGRPEN1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Enable |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:1]

Reserved, RES0.

Enable, bit [0]

Enables Group 1 interrupts for the current Security state.

| Enable | Meaning |
|--------|---|
| 0b0 | Group 1 interrupts are disabled for the current Security state. |
| 0b1 | Group 1 interrupts are enabled for the current Security state. |

Virtual accesses to this register update [ICH_VMCR_EL2.VENG1](#).

If EL3 is present:

- The Secure [ICC_IGRPEN1_EL1.Enable](#) bit is a read/write alias of the [ICC_IGRPEN1_EL3.EnableGrp1S](#) bit.
- The Non-secure [ICC_IGRPEN1_EL1.Enable](#) bit is a read/write alias of the [ICC_IGRPEN1_EL3.EnableGrp1NS](#) bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_IGRPEN1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_IGRPEN1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ICC_IGRPENn_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        return ICC_IGRPEN1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_IGRPEN1_EL1_S;
        else
            return ICC_IGRPEN1_EL1_NS;
    else
        return ICC_IGRPEN1_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_IGRPEN1_EL1_S;
        else
            return ICC_IGRPEN1_EL1_NS;
    else
        return ICC_IGRPEN1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            return ICC_IGRPEN1_EL1_S;
        else
            return ICC_IGRPEN1_EL1_NS;

```

MSR ICC_IGRPEN1_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ICC_IGRPENn_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        ICV_IGRPEN1_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_IGRPEN1_EL1_S = X[t];
        else
            ICC_IGRPEN1_EL1_NS = X[t];
    else
        ICC_IGRPEN1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_IGRPEN1_EL1_S = X[t];
        else
            ICC_IGRPEN1_EL1_NS = X[t];
    else
        ICC_IGRPEN1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_IGRPEN1_EL1_S = X[t];
        else
            ICC_IGRPEN1_EL1_NS = X[t];

```

ICC_IGRPEN1_EL3, Interrupt Controller Interrupt Group 1 Enable register (EL3)

The ICC_IGRPEN1_EL3 characteristics are:

Purpose

Controls whether Group 1 interrupts are enabled or not.

Configuration

AArch64 System register ICC_IGRPEN1_EL3 bits [31:0] can be mapped to AArch32 System register [ICC_MGRPEN1\[31:0\]](#), but this is not architecturally mandated.

This register is present only when FEAT_GICv3 is implemented and EL3 is implemented. Otherwise, direct accesses to ICC_IGRPEN1_EL3 are UNDEFINED.

Attributes

ICC_IGRPEN1_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|----|-------------|----|--|--|--------------|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | | 33 | | 32 | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | EnableGrp1S | | | | EnableGrp1NS | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | | 1 | | 0 | | | | | | |

Bits [63:2]

Reserved, RES0.

EnableGrp1S, bit [1]

Enables Group 1 interrupts for the Secure state.

| EnableGrp1S | Meaning |
|-------------|---|
| 0b0 | Secure Group 1 interrupts are disabled. |
| 0b1 | Secure Group 1 interrupts are enabled. |

The Secure [ICC_IGRPEN1_EL1](#).Enable bit is a read/write alias of the ICC_IGRPEN1_EL3.EnableGrp1S bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

EnableGrp1NS, bit [0]

Enables Group 1 interrupts for the Non-secure state.

| EnableGrp1NS | Meaning |
|--------------|---|
| 0b0 | Non-secure Group 1 interrupts are disabled. |
| 0b1 | Non-secure Group 1 interrupts are enabled. |

The Non-secure [ICC_IGRPEN1_EL1](#).Enable bit is a read/write alias of the ICC_IGRPEN1_EL3.EnableGrp1NS bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_IGRPEN1_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_IGRPEN1_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1100 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IGRPEN1_EL3;

```

MSR ICC_IGRPEN1_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1100 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN1_EL3 = X[t];

```

ICC_NMIAR1_EL1, Interrupt Controller Non-maskable Interrupt Acknowledge Register 1

The ICC_NMIAR1_EL1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled Group 1 non-maskable interrupt. This read acts as an acknowledge for the interrupt.

Configuration

This register is present only when FEAT_GICv3_NMI is implemented. Otherwise, direct accesses to ICC_NMIAR1_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when $PSTATE.\{I,F\} == \{0,0\}$). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_NMIAR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

This is the INTID of the highest priority pending interrupt, if that interrupt has the Non-maskable property and is of sufficient priority for it to be signalled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR_EL1.IDbits](#) and [ICC_CTLR_EL3.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_NMIAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_NMIAR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        return ICV_NMIAR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_NMIAR1_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_NMIAR1_EL1;
        elsif PSTATE.EL == EL3 then
            if ICC_SRE_EL3.SRE == '0' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_NMIAR1_EL1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_PMR_EL1, Interrupt Controller Interrupt Priority Mask Register

The ICC_PMR_EL1 characteristics are:

Purpose

- Provides an interrupt priority filter. Only interrupts with a higher priority than the value in this register are signaled to the PE.
- Writes to this register must be high performance and must ensure that no interrupt of lower priority than the written value occurs after the write, without requiring an ISB or an exception boundary.

Configuration

- AArch64 System register ICC_PMR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ICC_PMR\[31:0\]](#).
- This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_PMR_EL1 are UNDEFINED.
- To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that writes to this register are self-synchronising. This ensures that no interrupts below the written PMR value will be taken after a write to this register is architecturally executed. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_PMR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | Priority | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:8]

Reserved, RES0.

Priority, bits [7:0]

- The priority mask level for the CPU interface. If the priority of an interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.
- The possible priority field values are as follows:

| Implemented priority bits | Possible priority field values | Number of priority levels |
|---------------------------|-------------------------------------|---------------------------|
| [7:0] | 0x00-0xFF (0-255), all values | 256 |
| [7:1] | 0x00-0xFE (0-254), even values only | 128 |
| [7:2] | 0x00-0xFC (0-252), in steps of 4 | 64 |
| [7:3] | 0x00-0xF8 (0-248), in steps of 8 | 32 |
| [7:4] | 0x00-0xF0 (0-240), in steps of 16 | 16 |

Unimplemented priority bits are RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_PMR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_PMR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        return ICV_PMR_EL1;
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        return ICV_PMR_EL1;
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_PMR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_PMR_EL1;
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_PMR_EL1;

```

MSR ICC_PMR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        ICV_PMR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        ICV_PMR_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_PMR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_PMR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_PMR_EL1 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_RPR_EL1, Interrupt Controller Running Priority Register

The ICC_RPR_EL1 characteristics are:

Purpose

Indicates the Running priority of the CPU interface.

Configuration

AArch64 System register ICC_RPR_EL1 performs the same function as AArch32 System register [ICC_RPR](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_RPR_EL1 are UNDEFINED.

Attributes

ICC_RPR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|--------|----|----|----|----|----|----|----|----|----|----|------|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| NMI | NMI_NS | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | Priority | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

NMI, bit [63]

When FEAT_GICv3_NMI is implemented:

Indicates whether the running priority is from a NMI.

| NMI | Meaning |
|-----|---|
| 0b0 | When GICD_CTLR.DS ==1, there are no Active NMIs, or all Active NMIs have undergone priority drop. When GICD_CTLR.DS ==0: <ul style="list-style-type: none"> For Non-secure and Realm reads, there are no Active Non-secure Group 1 NMIs, or all Active Non-secure Group 1 NMIs have undergone priority drop. For Secure and Root reads, there are no Active Secure Group 1 NMIs, or all Active Secure Group 1 NMIs have undergone priority drop. |
| 0b1 | When GICD_CTLR.DS ==1, there is an Active NMI. When GICD_CTLR.DS ==0: <ul style="list-style-type: none"> For Non-secure and Realm reads, there is an Active Non-secure Group 1 NMIs. For Secure and Root reads, there is an Active Secure Group 1 NMIs. |

Otherwise:

Reserved, RES0.

NMI_NS, bit [62]**When FEAT_GICv3_NMI is implemented and EL3 is implemented:**

Indicates whether the running priority is from a Non-secure NMI.

| NMI_NS | Meaning |
|---------------|--|
| 0b0 | There are no Active Non-secure Group 1 NMIs, or all Active Non-secure Group 1 NMIs have undergone priority drop. |
| 0b1 | There is an Active Non-secure Group 1 NMI which has not undergone priority drop. |

Otherwise:

Reserved, RES0.

Bits [61:8]

Reserved, RES0.

Priority, bits [7:0]

The current running priority on the CPU interface. This is the group priority of the current active interrupt.

If there are no active interrupts on the CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

The priority returned is the group priority as if the BPR for the current Exception level and Security state was set to the minimum value of BPR for the number of implemented priority bits.

Note

If 8 bits of priority are implemented the group priority is bits[7:1] of the priority.

Accessing ICC_RPR_EL1

Software cannot determine the number of implemented priority bits from a read of this register.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_RPR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b000 | 0b1100 | 0b1011 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        return ICV_RPR_EL1;
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        return ICV_RPR_EL1;
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_RPR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_RPR_EL1;
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_RPR_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_SGI0R_EL1, Interrupt Controller Software Generated Interrupt Group 0 Register

The ICC_SGI0R_EL1 characteristics are:

Purpose

Generates Secure Group 0 SGIs.

Configuration

AArch64 System register ICC_SGI0R_EL1 performs the same function as AArch32 System register [ICC_SGI0R](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_SGI0R_EL1 are UNDEFINED.

Attributes

ICC_SGI0R_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|----|----|------|----|----|----|----|----|----|----|------------|----|----|----|------|----|-----|------|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | Aff3 | | | | | | | | RS | | | | RES0 | | IRM | Aff2 | | | | | | | | |
| RES0 | | | | INTID | | | | Aff1 | | | | | | | | TargetList | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:56]

Reserved, RES0.

Aff3, bits [55:48]

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

RS, bits [47:44]

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value $((RS * 16) + n)$.

When [ICC_CTLR_EL1](#).RSS==0, RS is RES0.

When [ICC_CTLR_EL1](#).RSS==1 and [GICD_TYPER](#).RSS==0, writing this register with $RS \neq 0$ is a CONSTRAINED UNPREDICTABLE choice of :

- The write is ignored.
- The RS field is treated as 0.

Bits [43:41]

Reserved, RES0.

IRM, bit [40]

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

| IRM | Meaning |
|-----|---|
| 0b0 | Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.<target list>. |
| 0b1 | Interrupts routed to all PEs in the system, excluding "self". |

Aff2, bits [39:32]

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]

Reserved, RES0.

INTID, bits [27:24]

The INTID of the SGI.

Aff1, bits [23:16]

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Note

If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

Accessing ICC_SGI0R_EL1

This register allows software executing in a Secure state to generate Group 0 SGIs. It will also allow software executing in a Non-secure state to generate Group 0 SGIs, if permitted by the settings of [GICR_NSACR](#) in the Redistributor corresponding to the target PE.

When [GICD_CTLR.DS](#)=0, Non-secure writes do not generate an interrupt for a target PE if not permitted by the [GICR_NSACR](#) register associated with the target PE. For more information, see 'Use of control registers for SGI forwarding' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Note

Accesses at EL3 are treated as Secure regardless of the value of SCR_EL3.NS.

Accesses to this register use the following encodings in the System register encoding space:

MSR ICC_SGI0R_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1011 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_SGI0R_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                ICC_SGI0R_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_SGI0R_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_SGI1R_EL1, Interrupt Controller Software Generated Interrupt Group 1 Register

The ICC_SGI1R_EL1 characteristics are:

Purpose

Generates Group 1 SGIs for the current Security state.

Configuration

AArch64 System register ICC_SGI1R_EL1 performs the same function as AArch32 System register [ICC_SGI1R](#).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_SGI1R_EL1 are UNDEFINED.

Under certain conditions a write to ICC_SGI1R_EL1 can generate Group 0 interrupts, see 'Forwarding an SGI to a target PE' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_SGI1R_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|----|----|------|----|----|----|----|----|----|----|------------|----|----|----|------|----|-----|------|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | Aff3 | | | | | | | | RS | | | | RES0 | | IRM | Aff2 | | | | | | | | |
| RES0 | | | | INTID | | | | Aff1 | | | | | | | | TargetList | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:56]

Reserved, RES0.

Aff3, bits [55:48]

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

RS, bits [47:44]

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value $((RS * 16) + n)$.

When [ICC_CTLR_EL1](#).RSS==0, RS is RES0.

When [ICC_CTLR_EL1](#).RSS==1 and [GICD_TYPER](#).RSS==0, writing this register with $RS \neq 0$ is a CONSTRAINED UNPREDICTABLE choice of :

- The write is ignored.
- The RS field is treated as 0.

Bits [43:41]

Reserved, RES0.

IRM, bit [40]

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

| IRM | Meaning |
|-----|---|
| 0b0 | Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.<target list>. |
| 0b1 | Interrupts routed to all PEs in the system, excluding "self". |

Aff2, bits [39:32]

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]

Reserved, RES0.

INTID, bits [27:24]

The INTID of the SGI.

Aff1, bits [23:16]

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Note

If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

Accessing ICC_SGI1R_EL1**Note**

Accesses at EL3 are treated as Secure regardless of the value of SCR_EL3.NS.

Accesses to this register use the following encodings in the System register encoding space:

MSR ICC_SGI1R_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1011 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_SGI1R_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_SGI1R_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_SGI1R_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_SRE_EL1, Interrupt Controller System Register Enable register (EL1)

The ICC_SRE_EL1 characteristics are:

Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL1.

Configuration

AArch64 System register ICC_SRE_EL1 bits [31:0] (S) are architecturally mapped to AArch32 System register [ICC_SRE\[31:0\]](#) (S).

AArch64 System register ICC_SRE_EL1 bits [31:0] (NS) are architecturally mapped to AArch32 System register [ICC_SRE\[31:0\]](#) (NS).

This register is present only when FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_SRE_EL1 are UNDEFINED.

Attributes

ICC_SRE_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:3]

Reserved, RES0.

DIB, bit [2]

Disable IRQ bypass.

| DIB | Meaning |
|-----|----------------------|
| 0b0 | IRQ bypass enabled. |
| 0b1 | IRQ bypass disabled. |

If EL3 is implemented and [GICD_CTLR](#).DS == 0, this field is a read-only alias of [ICC_SRE_EL3](#).DIB.

If EL3 is implemented and [GICD_CTLR](#).DS == 1, and EL2 is not implemented, this field is a read-write alias of [ICC_SRE_EL3](#).DIB.

If EL3 is not implemented and EL2 is implemented, this field is a read-only alias of [ICC_SRE_EL2](#).DIB.

If [GICD_CTLR](#).DS == 1 and EL2 is implemented, this field is a read-only alias of [ICC_SRE_EL2](#).DIB.

In systems that do not support IRQ bypass, this field is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

DFB, bit [1]

Disable FIQ bypass.

| DFB | Meaning |
|-----|----------------------|
| 0b0 | FIQ bypass enabled. |
| 0b1 | FIQ bypass disabled. |

If EL3 is implemented and [GICD_CTLR.DS](#) == 0, this field is a read-only alias of [ICC_SRE_EL3.DFB](#).

If EL3 is implemented and [GICD_CTLR.DS](#) == 1, and EL2 is not implemented, this field is a read-write alias of [ICC_SRE_EL3.DFB](#).

If EL3 is not implemented and EL2 is implemented, this field is a read-only alias of [ICC_SRE_EL2.DFB](#).

If [GICD_CTLR.DS](#) == 1 and EL2 is implemented, this field is a read-only alias of [ICC_SRE_EL2.DFB](#).

In systems that do not support FIQ bypass, this field is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

SRE, bit [0]

System Register Enable.

| SRE | Meaning |
|-----|--|
| 0b0 | The memory-mapped interface must be used. Access at EL1 to any ICC_* System register other than ICC_SRE_EL1 is trapped to EL1. |
| 0b1 | The System register interface for the current Security state is enabled. |

If software changes this bit from 1 to 0 in the Secure instance of this register, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

If EL3 is implemented and [ICC_SRE_EL3.SRE](#)==0 the Secure copy of this bit is RAZ/WI. If [ICC_SRE_EL3.SRE](#) is changed from zero to one, the Secure copy of this bit becomes UNKNOWN.

If EL2 is implemented and [ICC_SRE_EL2.SRE](#)==0 the Non-secure copy of this bit is RAZ/WI. If [ICC_SRE_EL2.SRE](#) is changed from zero to one, the Non-secure copy of this bit becomes UNKNOWN.

If EL3 is implemented and [ICC_SRE_EL3.SRE](#)==0 the Non-secure copy of this bit is RAZ/WI. If [ICC_SRE_EL3.SRE](#) is changed from zero to one, the Non-secure copy of this bit becomes UNKNOWN.

If Realm Management Extension is implemented, this field is RAO/WI.

GICv3 implementations that do not require GICv2 compatibility might choose to make this bit RAO/WI. The following options are supported:

- The Non-secure copy of [ICC_SRE_EL1.SRE](#) can be RAO/WI if [ICC_SRE_EL2.SRE](#) is also RAO/WI. This means all Non-secure software, including VMs using only virtual interrupts, must access the GIC using System registers.
- The Secure copy of [ICC_SRE_EL1.SRE](#) can be RAO/WI if [ICC_SRE_EL3.SRE](#) and [ICC_SRE_EL2.SRE](#) are also RAO/WI. This means that all Secure software must access the GIC using System registers and all Non-secure accesses to registers for physical interrupts must use System registers.

Note

A VM using only virtual interrupts might still use memory-mapped access if the Non-secure copy of [ICC_SRE_EL1.SRE](#) is not RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_SRE_EL1

Execution with [ICC_SRE_EL1](#).SRE set to 0 might make some System registers UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_SRE_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif EL2Enabled() && ICC_SRE_EL2.Enable == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && ICC_SRE_EL3.Enable == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_SRE_EL1_S;
        else
            return ICC_SRE_EL1_NS;
    else
        return ICC_SRE_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && ICC_SRE_EL3.Enable == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_SRE_EL1_S;
        else
            return ICC_SRE_EL1_NS;
    else
        return ICC_SRE_EL1;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        return ICC_SRE_EL1_S;
    else
        return ICC_SRE_EL1_NS;

```

MSR ICC_SRE_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif EL2Enabled() && ICC_SRE_EL2.Enable == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && ICC_SRE_EL3.Enable == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_SRE_EL1_S = X[t];
        else
            ICC_SRE_EL1_NS = X[t];
    else
        ICC_SRE_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && ICC_SRE_EL3.Enable == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_SRE_EL1_S = X[t];
        else
            ICC_SRE_EL1_NS = X[t];
    else
        ICC_SRE_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        ICC_SRE_EL1_S = X[t];
    else
        ICC_SRE_EL1_NS = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_SRE_EL2, Interrupt Controller System Register Enable register (EL2)

The ICC_SRE_EL2 characteristics are:

Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL2.

Configuration

AArch64 System register ICC_SRE_EL2 is architecturally mapped to AArch32 System register [ICC_HSRE](#).

This register is present only when FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICC_SRE_EL2 are UNDEFINED.

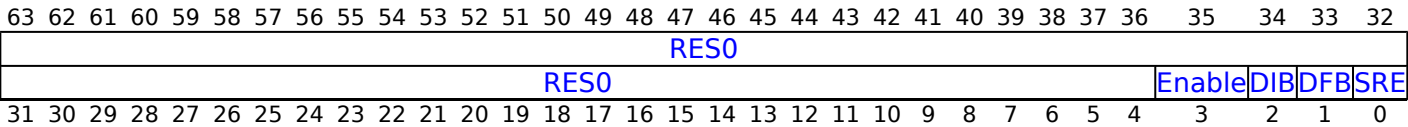
If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICC_SRE_EL2 is a 64-bit register.

Field descriptions



Bits [63:4]

Reserved, RES0.

Enable, bit [3]

Enable. Enables lower Exception level access to [ICC_SRE_EL1](#).

| Enable | Meaning |
|--------|---|
| 0b0 | When EL2 is implemented and enabled in the current Security state, EL1 accesses to ICC_SRE_EL1 trap to EL2. |
| 0b1 | EL1 accesses to ICC_SRE_EL1 do not trap to EL2. |

If ICC_SRE_EL2.SRE is RAO/WI, an implementation is permitted to make the Enable bit RAO/WI.

If ICC_SRE_EL2.SRE is 0, the Enable bit behaves as 1 for all purposes other than reading the value of the bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIB, bit [2]

Disable IRQ bypass.

| DIB | Meaning |
|-----|----------------------|
| 0b0 | IRQ bypass enabled. |
| 0b1 | IRQ bypass disabled. |

If EL3 is implemented and [GICD_CTLR.DS](#) is 0, this field is a read-only alias of [ICC_SRE_EL3.DIB](#).

If EL3 is implemented and [GICD_CTLR.DS](#) is 1, this field is a read-write alias of [ICC_SRE_EL3.DIB](#).

In systems that do not support IRQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

DFB, bit [1]

Disable FIQ bypass.

| DFB | Meaning |
|-----|----------------------|
| 0b0 | FIQ bypass enabled. |
| 0b1 | FIQ bypass disabled. |

If EL3 is implemented and [GICD_CTLR.DS](#) is 0, this field is a read-only alias of [ICC_SRE_EL3.DFB](#).

If EL3 is implemented and [GICD_CTLR.DS](#) is 1, this field is a read-write alias of [ICC_SRE_EL3.DFB](#).

In systems that do not support FIQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

SRE, bit [0]

System Register Enable.

| SRE | Meaning |
|-----|---|
| 0b0 | The memory-mapped interface must be used. Access at EL2 to any ICH_* or ICC_* register other than ICC_SRE_EL1 or ICC_SRE_EL2 , is trapped to EL2. |
| 0b1 | The System register interface to the ICH_* registers and the EL1 and EL2 ICC_* registers is enabled for EL2. |

If software changes this bit from 1 to 0, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

If EL3 is implemented and [ICC_SRE_EL3.SRE](#)==0 this bit is RAZ/WI. If [ICC_SRE_EL3.SRE](#) is changed from zero to one, this bit becomes UNKNOWN.

If Realm Management Extension is implemented, this field is RAO/WI.

FEAT_GICv3 implementations that do not require GICv2 compatibility might choose to make this bit RAO/WI, but this is only allowed if [ICC_SRE_EL3.SRE](#) is also RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_SRE_EL2

Execution with [ICC_SRE_EL2.SRE](#) set to 0 might make some System registers UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_SRE_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b1001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && ICC_SRE_EL3.Enable == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_SRE_EL2;
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        return ICC_SRE_EL2;

```

MSR ICC_SRE_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b1001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && ICC_SRE_EL3.Enable == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_SRE_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        ICC_SRE_EL2 = X[t];

```

ICC_SRE_EL3, Interrupt Controller System Register Enable register (EL3)

The ICC_SRE_EL3 characteristics are:

Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL3.

Configuration

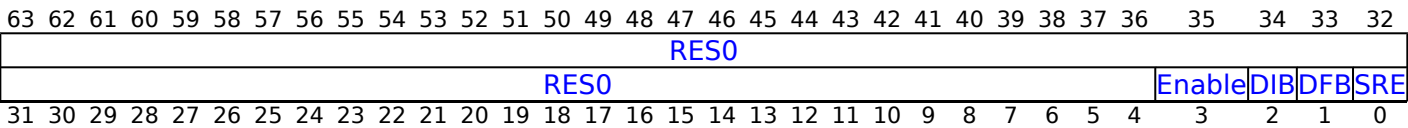
AArch64 System register ICC_SRE_EL3 bits [31:0] can be mapped to AArch32 System register [ICC_MSRE\[31:0\]](#), but this is not architecturally mandated.

This register is present only when FEAT_GICv3 is implemented and EL3 is implemented. Otherwise, direct accesses to ICC_SRE_EL3 are UNDEFINED.

Attributes

ICC_SRE_EL3 is a 64-bit register.

Field descriptions



Bits [63:4]

Reserved, RES0.

Enable, bit [3]

Enable. Enables lower Exception level access to [ICC_SRE_EL1](#) and [ICC_SRE_EL2](#).

| Enable | Meaning |
|--------|--|
| 0b0 | EL1 accesses to ICC_SRE_EL1 trap to EL3, unless these accesses are trapped to EL2 as a result of ICC_SRE_EL2.Enable == 0. EL2 accesses to ICC_SRE_EL1 and ICC_SRE_EL2 trap to EL3. |
| 0b1 | EL1 accesses to ICC_SRE_EL1 do not trap to EL3. EL2 accesses to ICC_SRE_EL1 and ICC_SRE_EL2 do not trap to EL3. |

If ICC_SRE_EL3.SRE is RAO/WI, an implementation is permitted to make the Enable bit RAO/WI.

If ICC_SRE_EL3.SRE is 0, the Enable bit behaves as 1 for all purposes other than reading the value of the bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIB, bit [2]

Disable IRQ bypass.

| DIB | Meaning |
|-----|----------------------|
| 0b0 | IRQ bypass enabled. |
| 0b1 | IRQ bypass disabled. |

In systems that do not support IRQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

DFB, bit [1]

Disable FIQ bypass.

| DFB | Meaning |
|-----|----------------------|
| 0b0 | FIQ bypass enabled. |
| 0b1 | FIQ bypass disabled. |

In systems that do not support FIQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

SRE, bit [0]

System Register Enable.

| SRE | Meaning |
|-----|--|
| 0b0 | The memory-mapped interface must be used. Access at EL3 to any ICH_* or ICC_* register other than ICC_SRE_EL1 , ICC_SRE_EL2 , or ICC_SRE_EL3 is trapped to EL3 |
| 0b1 | The System register interface to the ICH_* registers and the EL1, EL2, and EL3 ICC_* registers is enabled for EL3. |

If software changes this bit from 1 to 0, the results are UNPREDICTABLE.

If Realm Management Extension is implemented, this field is RAO/WI.

FEAT_GICv3 implementations that do not require GICv2 compatibility might choose to make this bit RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_SRE_EL3

This register is always System register accessible.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_SRE_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1100 | 0b1100 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return ICC_SRE_EL3;

```

MSR ICC_SRE_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1100 | 0b1100 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    ICC_SRE_EL3 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_AP0R<n>_EL2, Interrupt Controller Hyp Active Priorities Group 0 Registers, n = 0 - 3

The ICH_AP0R<n>_EL2 characteristics are:

Purpose

Provides information about Group 0 virtual active priorities for EL2.

Configuration

AArch64 System register ICH_AP0R<n>_EL2 bits [31:0] are architecturally mapped to AArch32 System register [ICH_AP0R<n>\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_AP0R<n>_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICH_AP0R<n>_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

P<x>, bit [x], for x = 31 to 0

Provides the access to the virtual active priorities for Group 0 interrupts. Possible values of each bit are:

| P<x> | Meaning |
|------|--|
| 0b0 | There is no Group 0 interrupt active with this priority level, or all active Group 0 interrupts with this priority level have undergone priority-drop. |
| 0b1 | There is a Group 0 interrupt active with this priority level which has not undergone priority drop. |

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP0R0_EL2 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP0R0_EL2 in the bits corresponding to 0:Priority[6:2].

- The active state of preemption levels 128 - 252 are held in ICH_AP0R1_EL2 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP0R0_EL2 in the bits corresponding to 00:Priority[5:1].
- The active state of preemption levels 64 - 126 are held in ICH_AP0R1_EL2 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP0R2_EL2 in the bits corresponding to 10:Priority[5:1].
- The active state of preemption levels 192 - 254 are held in ICH_AP0R3_EL2 in the bits corresponding to 11:Priority[5:1].

Note

Having the bit corresponding to a priority set to 1 in both ICH_AP0R<n>_EL2 and [ICH_AP1R<n>_EL2](#) might result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Software must ensure that ICH_AP0R<n>_EL2 is 0 for legacy VMs otherwise behaviour is UNPREDICTABLE. For more information about support for legacy VMs, see 'Support for legacy operation of VMs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The active priorities for Group 0 and Group 1 interrupts for legacy VMs are held in [ICH_AP1R<n>_EL2](#) and reads and writes to GICV_APR access [ICH_AP1R<n>_EL2](#). This means that ICH_AP0R<n>_EL2 is inaccessible to legacy VMs.

Accessing ICH_AP0R<n>_EL2

ICH_AP0R1_EL2 is only implemented in implementations that support 6 or more bits of preemption. ICH_AP0R2_EL2 and ICH_AP0R3_EL2 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by [ICH_VTR_EL2.PREbits](#)

Writing to these registers with any value other than the last read value of the register (or 0x00000000 for a newly set up virtual machine) can result in UNPREDICTABLE behavior of the virtual interrupt prioritization system allowing either:

- Virtual interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution at EL1 or EL0.

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICH_AP0R<n>_EL2.
- [ICH_AP1R<n>_EL2](#).

Having the bit corresponding to a priority set in both ICH_AP0R<n>_EL2 and [ICH_AP1R<n>_EL2](#) can result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICH_AP0R<n>_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|------------|
| 0b11 | 0b100 | 0b1100 | 0b1000 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x480+8*UInt(op2<1:0>)];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_AP0R_EL2[UInt(op2<1:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_AP0R_EL2[UInt(op2<1:0>)];

```

MSR ICH_AP0R<n>_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|------------|
| 0b11 | 0b100 | 0b1100 | 0b1000 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x480+8*UInt(op2<1:0>)] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_AP0R_EL2[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_AP0R_EL2[UInt(op2<1:0>)] = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_AP1R<n>_EL2, Interrupt Controller Hyp Active Priorities Group 1 Registers, n = 0 - 3

The ICH_AP1R<n>_EL2 characteristics are:

Purpose

Provides information about Group 1 virtual active priorities for EL2.

Configuration

AArch64 System register ICH_AP1R<n>_EL2 bits [31:0] are architecturally mapped to AArch32 System register [ICH_AP1R<n>\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_AP1R<n>_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICH_AP1R<n>_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| NMI | RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

NMI, bit [63]

When FEAT_GICv3_NMI is implemented:

Indicates whether the running virtual priority is from a NMI.

| NMI | Meaning |
|-----|--|
| 0b0 | There is no active Group 1 NMI, or all active Group 1 NMIs have undergone priority drop. |
| 0b1 | There is an active Group 1 NMI. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [62:32]

Reserved, RES0.

P<x>, bit [x], for x = 31 to 0

Group 1 interrupt active priorities. Possible values of each bit are:

| P<x> | Meaning |
|-------------------|--|
| 0b0 | There is no Group 1 interrupt active with this priority level, or all active Group 1 interrupts with this priority level have undergone priority-drop. |
| 0b1 | There is a Group 1 interrupt active with this priority level which has not undergone priority drop. |

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP1R0_EL2 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP1R0_EL2 in the bits corresponding to 0:Priority[6:2].
- The active state of preemption levels 128 - 252 are held in ICH_AP1R1_EL2 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP1R0_EL2 in the bits corresponding to 00:Priority[5:1].
- The active state of preemption levels 64 - 126 are held in ICH_AP1R1_EL2 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP1R2_EL2 in the bits corresponding to 10:Priority[5:1].
- The active state of preemption levels 192 - 254 are held in ICH_AP1R3_EL2 in the bits corresponding to 11:Priority[5:1].

Note

Having the bit corresponding to a priority set to 1 in both [ICH_AP0R<n>_EL2](#) and ICH_AP1R<n>_EL2 might result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

This register is always used for legacy VMs, regardless of the group of the virtual interrupt. Reads and writes to [GICV_APR<n>](#) access [ICH_AP1R<n>_EL2](#). For more information about support for legacy VMs, see 'Support for legacy operation of VMs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Accessing ICH_AP1R<n>_EL2

ICH_AP1R1_EL2 is only implemented in implementations that support 6 or more bits of preemption. ICH_AP1R2_EL2 and ICH_AP1R3_EL2 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by [ICH_VTR_EL2](#).PREbits

Writing to these registers with any value other than the last read value of the register (or 0x00000000 for a newly set up virtual machine) can result in UNPREDICTABLE behavior of the virtual interrupt prioritization system allowing either:

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICH_AP1R<n>_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|------------|
| 0b11 | 0b100 | 0b1100 | 0b1001 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x4A0+8*UInt(op2<1:0>)];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_AP1R_EL2[UInt(op2<1:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_AP1R_EL2[UInt(op2<1:0>)];

```

MSR ICH_AP1R<n>_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|------------|
| 0b11 | 0b100 | 0b1100 | 0b1001 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x4A0+8*UInt(op2<1:0>)] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_AP1R_EL2[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_AP1R_EL2[UInt(op2<1:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_EISR_EL2, Interrupt Controller End of Interrupt Status Register

The ICH_EISR_EL2 characteristics are:

Purpose

Indicates which List registers have outstanding EOI maintenance interrupts.

Configuration

AArch64 System register ICH_EISR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [ICH_EISR\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_EISR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICH_EISR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----------|----------|----------|----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| RES0 | | | | | | | | | | | | | | | | Status15 | Status14 | Status13 | Status12 | Status11 | Status10 | Status9 | Status8 | Status7 | Status6 | Status5 | Status4 | Status3 | Status2 | Status1 | Status0 | RES0 | | | | | | | | | | | | | | | |

Bits [63:16]

Reserved, RES0.

Status<n>, bit [n], for n = 15 to 0

EOI maintenance interrupt status bit for List register <n>:

| Status<n> | Meaning |
|-----------|--|
| 0b0 | List register <n>, ICH_LR<n>_EL2 , does not have an EOI maintenance interrupt. |
| 0b1 | List register <n>, ICH_LR<n>_EL2 , has an EOI maintenance interrupt that has not been handled. |

For any [ICH_LR<n>_EL2](#), the corresponding status bit is set to 1 if all of the following are true:

- [ICH_LR<n>_EL2](#).State is 0b00.
- [ICH_LR<n>_EL2](#).HW is 0.
- [ICH_LR<n>_EL2](#).EOI (bit [41]) is 1, indicating that when the interrupt corresponding to that List register is deactivated, a maintenance interrupt is asserted.

Otherwise the status bit takes the value 0.

Accessing ICH_EISR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICH_EISR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_EISR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_EISR_EL2;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_ELRSR_EL2, Interrupt Controller Empty List Register Status Register

The ICH_ELRSR_EL2 characteristics are:

Purpose

These registers can be used to locate a usable List register when the hypervisor is delivering an interrupt to a VM.

Configuration

AArch64 System register ICH_ELRSR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [ICH_ELRSR\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_ELRSR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICH_ELRSR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | |
|----------------------------------|----------|----------|----------|----------|----------|----------|---------|---------|---------|---------|
| 63626160595857565554535251504948 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 |
| RES0 | | | | | | | | | | |
| RES0 | Status15 | Status14 | Status13 | Status12 | Status11 | Status10 | Status9 | Status8 | Status7 | Status6 |
| 31302928272625242322212019181716 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 |

Bits [63:16]

Reserved, RES0.

Status<n>, bit [n], for n = 15 to 0

Status bit for List register <n>, [ICH LR<n> EL2](#):

| Status<n> | Meaning |
|-----------|--|
| 0b0 | List register ICH_LR<n>_EL2 , if implemented, contains a valid interrupt. Using this List register can result in overwriting a valid interrupt. |
| 0b1 | List register ICH_LR<n>_EL2 does not contain a valid interrupt. The List register is empty and can be used without overwriting a valid interrupt or losing an EOI maintenance interrupt. |

For any List register <n>, the corresponding status bit is set to 1 if [ICH_LR<n>_EL2.State](#) is 0b00 and either [ICH_LR<n>_EL2.HW](#) is 1 or [ICH_LR<n>_EL2.EOI](#) (bit [41]) is 0.

Otherwise the status bit takes the value 0.

Accessing ICH ELRSR EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICH_ELRSR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_ELRSR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_ELRSR_EL2;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_HCR_EL2, Interrupt Controller Hyp Control Register

The ICH_HCR_EL2 characteristics are:

Purpose

Controls the environment for VMs.

Configuration

AArch64 System register ICH_HCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [ICH_HCR\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_HCR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICH_HCR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|------|----|----|----|------|------|------|-------|-------|----|------|--------------|----------|----------|----------|----------|----|----|----|----|----|----|----|----|----|---|---|---|---|---|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| EOIcount | | RES0 | | | | DVIM | TDIR | TSEI | TALL1 | TALL0 | TC | RES0 | vSGIEOICount | VGrp1DIE | VGrp1EIE | VGrp0DIE | VGrp0EIE | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

EOIcount, bits [31:27]

This field is incremented whenever a successful write to a virtual EOIR or DIR register would have resulted in a virtual interrupt deactivation. That is either:

- A virtual write to EOIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when EOI mode is zero and no List Register was found.
- A virtual write to DIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when EOI mode is one and no List Register was found.

This allows software to manage more active interrupts than there are implemented List Registers.

It is CONSTRAINED UNPREDICTABLE whether a virtual write to EOIR that does not clear a bit in the Active Priorities registers ([ICH_AP0R<n>_EL2](#)/[ICH_AP1R<n>_EL2](#)) increments EOIcount. Permitted behaviors are:

- Increment EOIcount.
- Leave EOIcount unchanged.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [26:16]

Reserved, RES0.

DVIM, bit [15]

When `ICH_VTR_EL2.DVIM == 1`:

Directly-injected Virtual Interrupt Mask.

| DVIM | Meaning |
|------|---|
| 0b0 | This control has no effect on the signalling of virtual interrupts. |
| 0b1 | Virtual interrupts received via direct-injection are not presented to the virtual CPU interface and not considered when determining the highest priority pending virtual interrupt. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

TDIR, bit [14]

When `FEAT_GICv3_TDIR` is implemented:

Trap EL1 writes to [ICC_DIR_EL1](#) and [ICV_DIR_EL1](#).

| TDIR | Meaning |
|------|--|
| 0b0 | EL1 writes of ICC_DIR_EL1 and ICV_DIR_EL1 are not trapped to EL2, unless trapped by other mechanisms. |
| 0b1 | EL1 writes of ICV_DIR_EL1 are trapped to EL2. It is IMPLEMENTATION DEFINED whether writes of ICC_DIR_EL1 are trapped. Not trapping ICC_DIR_EL1 writes is DEPRECATED. |

Arm deprecates not including this trap bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

TSEI, bit [13]

Trap all locally generated SEIs. This bit allows the hypervisor to intercept locally generated SEIs that would otherwise be taken at EL1.

| TSEI | Meaning |
|------|--|
| 0b0 | Locally generated SEIs do not cause a trap to EL2. |
| 0b1 | Locally generated SEIs trap to EL2. |

If [ICH_VTR_EL2](#).SEIS is 0, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

TALL1, bit [12]

Trap all EL1 accesses to ICC_* and ICV_* System registers for Group 1 interrupts to EL2.

| TALL1 | Meaning |
|--------------|---|
| 0b0 | EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts proceed as normal. |
| 0b1 | EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts trap to EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

TALL0, bit [11]

Trap all EL1 accesses to ICC_* and ICV_* System registers for Group 0 interrupts to EL2.

| TALL0 | Meaning |
|--------------|---|
| 0b0 | EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts proceed as normal. |
| 0b1 | EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts trap to EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

TC, bit [10]

Trap all EL1 accesses to System registers that are common to Group 0 and Group 1 to EL2.

| TC | Meaning |
|-----------|---|
| 0b0 | EL1 accesses to common registers proceed as normal. |
| 0b1 | EL1 accesses to common registers trap to EL2. |

This affects accesses to [ICC_SGI0R_EL1](#), [ICC_SGI1R_EL1](#), [ICC_ASGI1R_EL1](#), [ICC_CTLR_EL1](#), [ICC_DIR_EL1](#), [ICC_PMR_EL1](#), [ICC_RPR_EL1](#), [ICV_CTLR_EL1](#), [ICV_DIR_EL1](#), [ICV_PMR_EL1](#), and [ICV_RPR_EL1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bit [9]

Reserved, RES0.

vSGIEOICount, bit [8]

When FEAT_GICv4p1 is implemented:

Controls whether deactivation of virtual SGIs can increment ICH_HCR_EL2.EOICount

| vSGIEOICount | Meaning |
|---------------------|---|
| 0b0 | Deactivation of virtual SGIs can increment ICH_HCR_EL2.EOICount. |
| 0b1 | Deactivation of virtual SGIs does not increment ICH_HCR_EL2.EOICount. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

VGrp1DIE, bit [7]

VM Group 1 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is disabled:

| VGrp1DIE | Meaning |
|----------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when ICH_VMCR_EL2.VENG1 is 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp1EIE, bit [6]

VM Group 1 Enabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is enabled:

| VGrp1EIE | Meaning |
|----------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when ICH_VMCR_EL2.VENG1 is 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp0DIE, bit [5]

VM Group 0 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is disabled:

| VGrp0DIE | Meaning |
|----------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when ICH_VMCR_EL2.VENG0 is 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp0EIE, bit [4]

VM Group 0 Enabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is enabled:

| VGrp0EIE | Meaning |
|----------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when ICH_VMCR_EL2.VENG0 is 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

NPiE, bit [3]

No Pending Interrupt Enable. Enables the signaling of a maintenance interrupt when there are no List registers with the State field set to 0b01 (pending):

| NPIE | Meaning |
|-------------|---|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled while the List registers contain no interrupts in the pending state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

LRENPIE, bit [2]

List Register Entry Not Present Interrupt Enable. Enables the signaling of a maintenance interrupt while the virtual CPU interface does not have a corresponding valid List register entry for an EOI request:

| LRENPIE | Meaning |
|----------------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt is asserted while the EOICount field is not 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

UIE, bit [1]

Underflow Interrupt Enable. Enables the signaling of a maintenance interrupt when the List registers are empty, or hold only one valid entry:

| UIE | Meaning |
|------------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt is asserted if none, or only one, of the List register entries is marked as a valid interrupt. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

En, bit [0]

Enable. Global enable bit for the virtual CPU interface:

| En | Meaning |
|-----------|---|
| 0b0 | Virtual CPU interface operation disabled. |
| 0b1 | Virtual CPU interface operation enabled. |

When this field is set to 0:

- The virtual CPU interface does not signal any maintenance interrupts.
- The virtual CPU interface does not signal any virtual interrupts.
- A read of [ICV_IAR0_EL1](#), [ICV_IAR1_EL1](#), [GICV_IAR](#) or [GICV_AIAR](#) returns a spurious interrupt ID.

Note

This field is RES0 when $SCR_EL3.\{NS,EEL2\} == \{0,0\}$

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICH_HCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICH_HCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x4C0];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_HCR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_HCR_EL2;

```

MSR ICH_HCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x4C0] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_HCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_HCR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_LR<n>_EL2, Interrupt Controller List Registers, n = 0 - 15

The ICH_LR<n>_EL2 characteristics are:

Purpose

Provides interrupt context information for the virtual CPU interface.

Configuration

AArch64 System register ICH_LR<n>_EL2 bits [31:0] are architecturally mapped to AArch32 System register [ICH_LR<n>\[31:0\]](#).

AArch64 System register ICH_LR<n>_EL2 bits [63:32] are architecturally mapped to AArch32 System register [ICH_LRC<n>\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_LR<n>_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

If list register n is not implemented, then accesses to this register are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICH_LR<n>_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|-------|-----|------|----------|----|----|----|----|----|----|----|----|----|------|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| State | HW | Group | NMI | RES0 | Priority | | | | | | | | | | RES0 | pINTID | | | | | | | | | | | | | | | | |
| vINTID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

State, bits [63:62]

The state of the interrupt:

| State | Meaning |
|-------|---------------------|
| 0b00 | Invalid (Inactive). |
| 0b01 | Pending. |
| 0b10 | Active. |
| 0b11 | Pending and active. |

The GIC updates these state bits as virtual interrupts proceed through the interrupt life cycle. Entries in the invalid state are ignored, except for the purpose of generating virtual maintenance interrupts.

For hardware interrupts, the pending and active state is held in the physical Distributor rather than the virtual CPU interface. A hypervisor must only use the pending and active state for software originated interrupts, which are typically associated with virtual devices, or SGIs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HW, bit [61]

Indicates whether this virtual interrupt maps directly to a hardware interrupt, meaning that it corresponds to a physical interrupt. Deactivation of the virtual interrupt also causes the deactivation of the physical interrupt with the ID that the pINTID field indicates.

| HW | Meaning |
|-----|---|
| 0b0 | The interrupt is triggered entirely by software. No notification is sent to the Distributor when the virtual interrupt is deactivated. |
| 0b1 | The interrupt maps directly to a hardware interrupt. A deactivate interrupt request is sent to the Distributor when the virtual interrupt is deactivated, using the pINTID field from this register to indicate the physical interrupt ID. If ICH_VMCR_EL2.VEOIM is 0, this request corresponds to a write to ICC_EOIR0_EL1 or ICC_EOIR1_EL1 . Otherwise, it corresponds to a write to ICC_DIR_EL1 . |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Group, bit [60]

Indicates the group for this virtual interrupt.

| Group | Meaning |
|-------|--|
| 0b0 | This is a Group 0 virtual interrupt. ICH_VMCR_EL2.VFIQEn determines whether it is signaled as a virtual IRQ or as a virtual FIQ, and ICH_VMCR_EL2.VENG0 enables signaling of this interrupt to the virtual machine. |
| 0b1 | This is a Group 1 virtual interrupt, signaled as a virtual IRQ. ICH_VMCR_EL2.VENG1 enables the signalling of this interrupt to the virtual machine. If ICH_VMCR_EL2.VCBPR is 0, then ICC_BPR1_EL1 determines if a pending Group 1 interrupt has sufficient priority to preempt current execution. Otherwise, ICH_LR<n>_EL2 determines preemption. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NMI, bit [59]

When FEAT_GICv3_NMI is implemented:

Indicates whether the virtual priority has the non-maskable property.

| NMI | Meaning |
|-----|---|
| 0b0 | vINTID does not have the non-maskable interrupt property. |
| 0b1 | vINTID has the non-maskable interrupt property. |

Setting ICH_LR<n>_EL2.NMI to 1 when ICH_LR<n>_EL2.State is not Invalid is CONSTRAINED UNPREDICTABLE if either ICH_LR<n>_EL2.vINTID indicates an LPI or ICH_LR<n>_EL2.Group is 0.

The permitted behaviours are:

- ICH_LR<n>_EL2.NMI is treated as 0 for all purposes other than a direct read of the register.
- The virtual interrupt is presented with superpriority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [58:56]

Reserved, RES0.

Priority, bits [55:48]

The priority of this interrupt.

It is IMPLEMENTATION DEFINED how many bits of priority are implemented, though at least five bits must be implemented. Unimplemented bits are RES0 and start from bit[48] up to bit[50]. The number of implemented bits can be discovered from [ICH_VTR_EL2.PRIBits](#).

When ICH_LR<n>_EL2.NMI is set to 1, this field is RES0 and the virtual interrupt's priority is treated as 0x00.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [47:45]

Reserved, RES0.

pINTID, bits [44:32]

Physical INTID, for hardware interrupts.

When ICH_LR<n>_EL2.HW is 0 (there is no corresponding physical interrupt), this field has the following meaning:

- Bits[44:42] : RES0.
- Bit[41] : EOI. If this bit is 1, then when the interrupt identified by vINTID is deactivated, a maintenance interrupt is asserted.
- Bits[40:32] : RES0.

When ICH_LR<n>_EL2.HW is 1 (there is a corresponding physical interrupt):

- This field indicates the physical INTID. This field is only required to implement enough bits to hold a valid value for the implemented INTID size. Any unused higher order bits are RES0.
- When [ICC_CTLR_EL1.ExtRange](#) is 0, then bits[44:42] of this field are RES0.
- If the value of pINTID is not a valid INTID, behavior is UNPREDICTABLE. If the value of pINTID indicates a PPI, this field applies to the PPI associated with this same physical PE ID as the virtual CPU interface requesting the deactivation.

A hardware physical identifier is only required in List Registers for interrupts that require deactivation. This means only 13 bits of Physical INTID are required, regardless of the number specified by [ICC_CTLR_EL1.IDbits](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

vINTID, bits [31:0]

Virtual INTID of the interrupt.

If the value of vINTID is 1020-1023 and ICH_LR<n>_EL2.State!=0b00 (Inactive), behavior is UNPREDICTABLE.

Behavior is UNPREDICTABLE if two or more List Registers specify the same vINTID when:

- ICH_LR<n>_EL2.State == 0b01.
- ICH_LR<n>_EL2.State == 0b10.
- ICH_LR<n>_EL2.State == 0b11.

It is IMPLEMENTATION DEFINED how many bits are implemented, though at least 16 bits must be implemented. Unimplemented bits are RES0. The number of implemented bits can be discovered from [ICH_VTR_EL2.IDbits](#).

When [ICC_SRE_EL1.SRE](#) == 0, specifying a vINTID in the LPI range is UNPREDICTABLE

Note

When a VM is using memory-mapped access to the GIC, software must ensure that the correct source PE ID is provided in bits[12:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICH_LR<n>_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICH_LR<n>_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|--------|
| 0b11 | 0b100 | 0b1100 | 0b110:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x400+8*UInt(CRm<0>:op2<2:0>)];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_LR_EL2[UInt(CRm<0>:op2<2:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_LR_EL2[UInt(CRm<0>:op2<2:0>)];

```

MSR ICH_LR<n>_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|--------|
| 0b11 | 0b100 | 0b1100 | 0b110:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x400+8*UInt(CRm<0>:op2<2:0>)] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_LR_EL2[UInt(CRm<0>:op2<2:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_LR_EL2[UInt(CRm<0>:op2<2:0>)] = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_MISR_EL2, Interrupt Controller Maintenance Interrupt State Register

The ICH_MISR_EL2 characteristics are:

Purpose

Indicates which maintenance interrupts are asserted.

Configuration

AArch64 System register ICH_MISR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [ICH_MISR\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_MISR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICH_MISR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|--------|----|--------|----|--------|----|--------|----|-----|----|-----|----|----|----|-----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | RES0 | | | | | | | | VGrp1D | | VGrp1E | | VGrp0D | | VGrp0E | | NPL | | REN | | U | | EOI | |

Bits [63:8]

Reserved, RES0.

VGrp1D, bit [7]

vPE Group 1 Disabled.

| VGrp1D | Meaning |
|--------|--|
| 0b0 | vPE Group 1 Disabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 1 Disabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR_EL2.VGrp1DIE](#)==1 and [ICH_VMCR_EL2.VENG1](#)==is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp1E, bit [6]

vPE Group 1 Enabled.

| VGrp1E | Meaning |
|--------|---|
| 0b0 | vPE Group 1 Enabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 1 Enabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR_EL2.VGrp1EIE==1](#) and [ICH_VMCR_EL2.VENG1==1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp0D, bit [5]

vPE Group 0 Disabled.

| VGrp0D | Meaning |
|--------|--|
| 0b0 | vPE Group 0 Disabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 0 Disabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR_EL2.VGrp0DIE==1](#) and [ICH_VMCR_EL2.VENG0==0](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp0E, bit [4]

vPE Group 0 Enabled.

| VGrp0E | Meaning |
|--------|---|
| 0b0 | vPE Group 0 Enabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 0 Enabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR_EL2.VGrp0EIE==1](#) and [ICH_VMCR_EL2.VENG0==1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

NP, bit [3]

No Pending.

| NP | Meaning |
|-----|--|
| 0b0 | No Pending maintenance interrupt not asserted. |
| 0b1 | No Pending maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR_EL2.NPIE==1](#) and no List register is in pending state.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

LREN, bit [2]

List Register Entry Not Present.

| LREN | Meaning |
|------|---|
| 0b0 | List Register Entry Not Present maintenance interrupt not asserted. |
| 0b1 | List Register Entry Not Present maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR_EL2.LRENPIE==1](#) and [ICH_HCR_EL2.EOIcount](#) is non-zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

U, bit [1]

Underflow.

| U | Meaning |
|-----|---|
| 0b0 | Underflow maintenance interrupt not asserted. |
| 0b1 | Underflow maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR_EL2.UIE==1](#) and zero or one of the List register entries are marked as a valid interrupt, that is, if the corresponding [ICH_LR<n>_EL2.State](#) bits do not equal 0x0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

EOI, bit [0]

End Of Interrupt.

| EOI | Meaning |
|-----|--|
| 0b0 | End Of Interrupt maintenance interrupt not asserted. |
| 0b1 | End Of Interrupt maintenance interrupt asserted. |

This maintenance interrupt is asserted when at least one bit in [ICH_EISR_EL2](#) is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

The U and NP bits do not include the status of any pending/active 'VSet (IRI)' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069) packets because these bits control generation of interrupts that allow software management of the contents of the List Registers (which are not affected by 'VSet (IRI)' packets).

Accessing ICH_MISR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICH_MISR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_MISR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_MISR_EL2;

```


ICH_VMCR_EL2, Interrupt Controller Virtual Machine Control Register

The ICH_VMCR_EL2 characteristics are:

Purpose

Enables the hypervisor to save and restore the virtual machine view of the GIC state.

Configuration

AArch64 System register ICH_VMCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [ICH_VMCR\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_VMCR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICH_VMCR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|-------|----|----|----|------|----|----|----|----|----|----|----|-------|----|----|----|------|----|----|----|-------|--|--|--|--------|--|--|--|---------|--|--|--|-------|--|--|--|-------|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| VPMR | | | | | | | | VBPR0 | | | | VBPR1 | | | | RES0 | | | | | | | | VEOIM | | | | RES0 | | | | VCBPR | | | | VFIQEn | | | | VAcKctI | | | | VENG1 | | | | VENG0 | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | | | | | | | |

Bits [63:32]

Reserved, RES0.

VPMR, bits [31:24]

Virtual Priority Mask. The priority mask level for the virtual CPU interface. If the priority of a pending virtual interrupt is higher than the value indicated by this field, the interface signals the virtual interrupt to the PE.

This field is an alias of [ICV_PMR_EL1](#).Priority.

VBPR0, bits [23:21]

Virtual Binary Point Register, Group 0. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption, and also determines Group 1 interrupt preemption if ICH_VMCR_EL2.VCBPR == 1.

This field is an alias of [ICV_BPR0_EL1](#).BinaryPoint.

The minimum value of this field is determined by [ICH_VTR_EL2](#).PREbits. An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value.

VBPR1, bits [20:18]

Virtual Binary Point Register, Group 1. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption if [ICH_VMCR_EL2.VCBPR](#) == 0.

This field is an alias of [ICV_BPR1_EL1.BinaryPoint](#).

This field is always accessible to EL2 accesses, regardless of the setting of the [ICH_VMCR_EL2.VCBPR](#) field.

For Non-secure writes, the minimum value of this field is the minimum value of [ICH_VMCR_EL2.VBPR0](#) plus one.

For Secure writes, the minimum value of this field is the minimum value of [ICH_VMCR_EL2.VBPR0](#).

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value.

Bits [17:10]

Reserved, RES0.

VEOIM, bit [9]

Virtual EOI mode. Controls whether a write to an End of Interrupt register also deactivates the virtual interrupt:

| VEOIM | Meaning |
|-------|---|
| 0b0 | ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR_EL1 are UNPREDICTABLE. |
| 0b1 | ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide priority drop functionality only. ICV_DIR_EL1 provides interrupt deactivation functionality. |

This bit is an alias of [ICV_CTLR_EL1.EOImode](#).

Bits [8:5]

Reserved, RES0.

VCBPR, bit [4]

Virtual Common Binary Point Register. Possible values of this bit are:

| VCBPR | Meaning |
|-------|---|
| 0b0 | ICV_BPR1_EL1 determines the preemption group for virtual Group 1 interrupts. |
| 0b1 | Reads of ICV_BPR1_EL1 return ICV_BPR0_EL1 plus one, saturated to 0b111. Writes to ICV_BPR1_EL1 are ignored. |

This field is an alias of [ICV_CTLR_EL1.CBPR](#).

VFIQEn, bit [3]

Virtual FIQ enable. Possible values of this bit are:

| VFIQEn | Meaning |
|--------|---|
| 0b0 | Group 0 virtual interrupts are presented as virtual IRQs. |
| 0b1 | Group 0 virtual interrupts are presented as virtual FIQs. |

This bit is an alias of [GICV_CTLR.FIQEn](#).

In implementations where the Non-secure copy of [ICC_SRE_EL1.SRE](#) is always 1, this bit is RES1.

VAckCtl, bit [2]

Virtual AckCtl. Possible values of this bit are:

| VAckCtl | Meaning |
|---------|--|
| 0b0 | If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns an INTID of 1022. |
| 0b1 | If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns the INTID of the corresponding interrupt. |

This bit is an alias of [GICV_CTLR](#).AckCtl.

This field is supported for backwards compatibility with GICv2. Arm deprecates the use of this field.

In implementations where the Non-secure copy of [ICC_SRE_EL1](#).SRE is always 1, this bit is RES0.

VENG1, bit [1]

Virtual Group 1 interrupt enable. Possible values of this bit are:

| VENG1 | Meaning |
|-------|--|
| 0b0 | Virtual Group 1 interrupts are disabled. |
| 0b1 | Virtual Group 1 interrupts are enabled. |

This bit is an alias of [ICV_IGRPEN1_EL1](#).Enable.

VENG0, bit [0]

Virtual Group 0 interrupt enable. Possible values of this bit are:

| VENG0 | Meaning |
|-------|--|
| 0b0 | Virtual Group 0 interrupts are disabled. |
| 0b1 | Virtual Group 0 interrupts are enabled. |

This bit is an alias of [ICV_IGRPEN0_EL1](#).Enable.

Accessing ICH_VMCR_EL2

When EL2 is using System register access, EL1 using either System register or memory-mapped access must be supported.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICH_VMCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x4C8];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_VMCR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_VMCR_EL2;

```

MSR ICH_VMCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x4C8] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        ICH_VMCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICH_VMCR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_VTR_EL2, Interrupt Controller VGIC Type Register

The ICH_VTR_EL2 characteristics are:

Purpose

Reports supported GIC virtualization features.

Configuration

AArch64 System register ICH_VTR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [ICH_VTR\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_VTR_EL2 are UNDEFINED.

If EL2 is not implemented, all bits in this register are RES0 from EL3, except for nV4, which is RES1 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

ICH_VTR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|---------|----|----|----|--------|----|----|----|------|-----|-----|-----|------|------|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PRibits | | | | PREbits | | | | IDbits | | | | SEIS | A3V | nV4 | TDS | DVIM | RES0 | | | | | | | | | | | | ListRegs | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:32]

Reserved, RES0.

PRibits, bits [31:29]

Priority bits. The number of virtual priority bits implemented, minus one.

An implementation must implement at least 32 levels of virtual priority (5 priority bits).

This field is an alias of [ICV_CTLR_EL1](#).PRibits.

PREbits, bits [28:26]

The number of virtual preemption bits implemented, minus one.

An implementation must implement at least 32 levels of virtual preemption priority (5 preemption bits).

The value of this field must be less than or equal to the value of ICH_VTR_EL2.PRibits.

The maximum value of this field is 6, indicating 7 bits of preemption.

This field determines the minimum value of [ICH_VMCR_EL2](#).VBPR0.

IDbits, bits [25:23]

The number of virtual interrupt identifier bits supported:

| IDbits | Meaning |
|--------|----------|
| 0b000 | 16 bits. |
| 0b001 | 24 bits. |

All other values are reserved.

This field is an alias of [ICV_CTLR_EL1](#).IDbits.

SEIS, bit [22]

SEI Support. Indicates whether the virtual CPU interface supports generation of SEIs:

| SEIS | Meaning |
|------|--|
| 0b0 | The virtual CPU interface logic does not support generation of SEIs. |
| 0b1 | The virtual CPU interface logic supports generation of SEIs. |

This bit is an alias of [ICV_CTLR_EL1](#).SEIS.

A3V, bit [21]

Affinity 3 Valid. Possible values are:

| A3V | Meaning |
|-----|---|
| 0b0 | The virtual CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers. |
| 0b1 | The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers. |

This bit is an alias of [ICV_CTLR_EL1](#).A3V.

nV4, bit [20]

Direct injection of virtual interrupts not supported. Possible values are:

| nV4 | Meaning |
|-----|--|
| 0b0 | The CPU interface logic supports direct injection of virtual interrupts. |
| 0b1 | The CPU interface logic does not support direct injection of virtual interrupts. |

In GICv3, the only permitted value is 0b1.

TDS, bit [19]

Separate trapping of EL1 writes to [ICV_DIR_EL1](#) supported.

| TDS | Meaning |
|-----|--|
| 0b0 | Implementation does not support ICH_HCR_EL2 .TDIR. |
| 0b1 | Implementation supports ICH_HCR_EL2 .TDIR. |

FEAT_GICv3_TDIR implements the functionality added by the value 0b1.

DVIM, bit [18]

Masking of directly-injected virtual interrupts.

| DVIM | Meaning |
|------|--|
| 0b0 | Masking of Directly-injected Virtual Interrupts not supported. |
| 0b1 | Masking of Directly-injected Virtual Interrupts is supported. |

When a PE implements the Realm Management Extension, this field is RAO/WI.

Bits [17:5]

Reserved, RES0.

ListRegs, bits [4:0]

The number of implemented List registers, minus one. For example, a value of 0b01111 indicates that the maximum of 16 List registers are implemented.

Accessing ICH_VTR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICH_VTR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ICH_VTR_EL2;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICH_VTR_EL2;
```

ICV_AP0R<n>_EL1, Interrupt Controller Virtual Active Priorities Group 0 Registers, n = 0 - 3

The ICV_AP0R<n>_EL1 characteristics are:

Purpose

Provides information about virtual Group 0 active priorities.

Configuration

AArch64 System register ICV_AP0R<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ICV_AP0R<n>\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_AP0R<n>_EL1 are UNDEFINED.

Attributes

ICV_AP0R<n>_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

Accessing ICV_AP0R<n>_EL1

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 0 active priorities) might result in UNPREDICTABLE behavior of the virtual interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICV_AP0R1_EL1 is only implemented in implementations that support 6 or more bits of priority. ICV_AP0R2_EL1 and ICV_AP0R3_EL1 are only implemented in implementations that support 7 bits of priority. Unimplemented registers are UNDEFINED.

Writing to the active priority registers in any order other than the following order might result in UNPREDICTABLE behavior of the interrupt prioritization system:

- ICV_AP0R<n>_EL1.
- [ICV_AP1R<n>_EL1](#).

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_AP0R<n>_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|------------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b1:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        return ICV_AP0R_EL1[UInt(op2<1:0>)];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_AP0R_EL1[UInt(op2<1:0>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_AP0R_EL1[UInt(op2<1:0>)];
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_AP0R_EL1[UInt(op2<1:0>)];

```

MSR ICC_AP0R<n>_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|------------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b1:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        ICV_AP0R_EL1[UInt(op2<1:0>)] = X[t];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_AP0R_EL1[UInt(op2<1:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_AP1R<n>_EL1, Interrupt Controller Virtual Active Priorities Group 1 Registers, n = 0 - 3

The ICV_AP1R<n>_EL1 characteristics are:

Purpose

Provides information about virtual Group 1 active priorities.

Configuration

AArch64 System register ICV_AP1R<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ICV_AP1R<n>\[31:0\]](#).
This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_AP1R<n>_EL1 are UNDEFINED.

Attributes

ICV_AP1R<n>_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| NMI | | RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

NMI, bit [63]
When FEAT_GICv3_NMI is implemented:

Indicates whether the running priority is from a NMI.

| NMI | Meaning |
|-----|--|
| 0b0 | There is no active Group 1 NMI, or all active Group 1 NMIs have undergone priority-drop. |
| 0b1 | There is an active Group 1 NMI. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [62:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

Accessing ICV_AP1R<n>_EL1

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 1 active priorities) might result in UNPREDICTABLE behavior of the virtual interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICV_AP1R1_EL1 is only implemented in implementations that support 6 or more bits of priority. ICV_AP1R2_EL1 and ICV_AP1R3_EL1 are only implemented in implementations that support 7 bits of priority. Unimplemented registers are UNDEFINED.

Writing to the active priority registers in any order other than the following order might result in UNPREDICTABLE behavior of the interrupt prioritization system:

- [ICV_AP0R<n>_EL1](#).
- ICV_AP1R<n>_EL1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_AP1R<n>_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|------------|
| 0b11 | 0b000 | 0b1100 | 0b1001 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        return ICV_AP1R_EL1[UInt(op2<1:0>)];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_AP1R_EL1_S[UInt(op2<1:0>)];
        else
            return ICC_AP1R_EL1_NS[UInt(op2<1:0>)];
    else
        return ICC_AP1R_EL1[UInt(op2<1:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_AP1R_EL1_S[UInt(op2<1:0>)];
        else
            return ICC_AP1R_EL1_NS[UInt(op2<1:0>)];
    else
        return ICC_AP1R_EL1[UInt(op2<1:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            return ICC_AP1R_EL1_S[UInt(op2<1:0>)];
        else
            return ICC_AP1R_EL1_NS[UInt(op2<1:0>)];

```

MSR ICC_AP1R<n>_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|------------|
| 0b11 | 0b000 | 0b1100 | 0b1001 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        ICV_AP1R_EL1[UInt(op2<1:0>)] = X[t];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[UInt(op2<1:0>)] = X[t];
        else
            ICC_AP1R_EL1_NS[UInt(op2<1:0>)] = X[t];
    else
        ICC_AP1R_EL1[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[UInt(op2<1:0>)] = X[t];
        else
            ICC_AP1R_EL1_NS[UInt(op2<1:0>)] = X[t];
    else
        ICC_AP1R_EL1[UInt(op2<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_AP1R_EL1_S[UInt(op2<1:0>)] = X[t];
        else
            ICC_AP1R_EL1_NS[UInt(op2<1:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_BPR0_EL1, Interrupt Controller Virtual Binary Point Register 0

The ICV_BPR0_EL1 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 0 interrupt preemption.

Configuration

AArch64 System register ICV_BPR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ICV_BPR0\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_BPR0_EL1 are UNDEFINED.

Attributes

ICV_BPR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | BinaryPoint | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. This is done as follows:

| Binary point value | Group priority field | Subpriority field | Field with binary point |
|--------------------|----------------------|-------------------|-------------------------|
| 0 | [7:1] | [0] | ggggggg.s |
| 1 | [7:2] | [1:0] | gggggg.ss |
| 2 | [7:3] | [2:0] | ggggg.sss |
| 3 | [7:4] | [3:0] | gggg.ssss |
| 4 | [7:5] | [4:0] | ggg.sssss |
| 5 | [7:6] | [5:0] | gg.ssssss |
| 6 | [7] | [6:0] | g.sssssss |
| 7 | No preemption | [7:0] | .ssssssss |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_BPR0_EL1

The minimum binary point value is derived from the number of implemented preemption bits, as shown in the following table:

| Number of implemented preemption bits | Minimum value of BPR0 |
|---------------------------------------|-----------------------|
| 7 | 0 |
| 6 | 1 |
| 5 | 2 |

The number of implemented preemption bits is indicated by [ICH_VTR_EL2](#).PREbits.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_BPR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        return ICV_BPR0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_BPR0_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_BPR0_EL1;
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_BPR0_EL1;

```

MSR ICC_BPR0_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        ICV_BPR0_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_BPR0_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_BPR1_EL1, Interrupt Controller Virtual Binary Point Register 1

The ICV_BPR1_EL1 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 1 interrupt preemption.

Configuration

AArch64 System register ICV_BPR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ICV_BPR1\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_BPR1_EL1 are UNDEFINED.

Attributes

ICV_BPR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | BinaryPoint | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

If the GIC is configured to use separate binary point fields for Group 0 and Group 1 interrupts, the value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

For more information about priorities, see 'Priority grouping' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

An attempt to program this field to a value less than the minimum value sets the field to the minimum value.

If [ICV_CTLR_EL1](#).CBPR is set to 1, Non-secure EL1 reads return [ICV_BPR0_EL1](#) + 1 saturated to 0b111. Non-secure EL1 writes are ignored.

If [ICV_CTLR_EL1](#).CBPR is set to 1, Secure EL1 reads return [ICV_BPR0_EL1](#). Secure EL1 writes modify [ICV_BPR0_EL1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_BPR1_EL1

For Non-secure writes, the minimum value of this field is the minimum value of [ICH_VMCR_EL2](#).VBPR0 plus one.

For Secure writes, the minimum value of this field is the minimum value of [ICH_VMCR_EL2.VBPR0](#).

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_BPR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        return ICV_BPR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_BPR1_EL1_S;
        else
            return ICC_BPR1_EL1_NS;
    else
        return ICC_BPR1_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_BPR1_EL1_S;
        else
            return ICC_BPR1_EL1_NS;
    else
        return ICC_BPR1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            return ICC_BPR1_EL1_S;
        else
            return ICC_BPR1_EL1_NS;

```

MSR ICC_BPR1_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b011 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_BPR1_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_BPR1_EL1_S = X[t];
        else
            ICC_BPR1_EL1_NS = X[t];
    else
        ICC_BPR1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_BPR1_EL1_S = X[t];
        else
            ICC_BPR1_EL1_NS = X[t];
    else
        ICC_BPR1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_BPR1_EL1_S = X[t];
        else
            ICC_BPR1_EL1_NS = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_CTLR_EL1, Interrupt Controller Virtual Control Register

The ICV_CTLR_EL1 characteristics are:

Purpose

Controls aspects of the behavior of the GIC virtual CPU interface and provides information about the features implemented.

Configuration

AArch64 System register ICV_CTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ICV_CTLR\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_CTLR_EL1 are UNDEFINED.

Attributes

ICV_CTLR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ExtRange | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |

Bits [63:20]

Reserved, RES0.

ExtRange, bit [19]

Extended INTID range (read-only).

| ExtRange | Meaning |
|---|--|
| 0b0 | CPU interface does not support INTIDs in the range 1024..8191. <ul style="list-style-type: none"> Behaviour is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface. |
| Note Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them. | |
| 0b1 | CPU interface supports INTIDs in the range 1024..8191 <ul style="list-style-type: none"> All INTIDs in the range 1024..8191 are treated as requiring deactivation. |

ICV_CTLR_EL1.ExtRange is an alias of [ICC_CTLR_EL1](#).ExtRange.

RSS, bit [18]

Range Selector Support. Possible values are:

| RSS | Meaning |
|------------|--|
| 0b0 | Targeted SGIs with affinity level 0 values of 0 - 15 are supported. |
| 0b1 | Targeted SGIs with affinity level 0 values of 0 - 255 are supported. |

This bit is read-only.

Bits [17:16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored. Possible values are:

| A3V | Meaning |
|------------|---|
| 0b0 | The virtual CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers. |
| 0b1 | The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers. |

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the virtual CPU interface supports local generation of SEIs:

| SEIS | Meaning |
|-------------|--|
| 0b0 | The virtual CPU interface logic does not support local generation of SEIs. |
| 0b1 | The virtual CPU interface logic supports local generation of SEIs. |

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. The number of virtual interrupt identifier bits supported:

| IDbits | Meaning |
|---------------|----------------|
| 0b000 | 16 bits. |
| 0b001 | 24 bits. |

All other values are reserved.

PRIbits, bits [10:8]

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation must implement at least 32 levels of physical priority (5 priority bits).

Note

This field always returns the number of priority bits implemented.

The division between group priority and subpriority is defined in the binary point registers [ICV_BPR0_EL1](#) and [ICV_BPR1_EL1](#).

Bits [7:2]

Reserved, RES0.

EOImode, bit [1]

Virtual EOI mode. Controls whether a write to an End of Interrupt register also deactivates the virtual interrupt:

| EOImode | Meaning |
|---------|---|
| 0b0 | ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR_EL1 are UNPREDICTABLE. |
| 0b1 | ICV_EOIR0_EL1 and ICV_EOIR1_EL1 provide priority drop functionality only. ICV_DIR_EL1 provides interrupt deactivation functionality. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CBPR, bit [0]

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both virtual Group 0 and virtual Group 1 interrupts:

| CBPR | Meaning |
|------|--|
| 0b0 | ICV_BPR1_EL1 determines the preemption group for virtual Group 1 interrupts. |
| 0b1 | Non-secure reads of ICV_BPR1_EL1 return ICV_BPR0_EL1 plus one, saturated to 0b111. Non-secure writes to ICV_BPR1_EL1 are ignored. Secure reads of ICV_BPR1_EL1 return ICV_BPR0_EL1 . Secure writes of ICV_BPR1_EL1 modify ICV_BPR0_EL1 . |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_CTLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_CTLR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        return ICV_CTLR_EL1;
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        return ICV_CTLR_EL1;
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_CTLR_EL1_S;
        else
            return ICC_CTLR_EL1_NS;
    else
        return ICC_CTLR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_CTLR_EL1_S;
        else
            return ICC_CTLR_EL1_NS;
    else
        return ICC_CTLR_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            return ICC_CTLR_EL1_S;
        else
            return ICC_CTLR_EL1_NS;

```

MSR ICC_CTLR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        ICV_CTLR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        ICV_CTLR_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t];
        else
            ICC_CTLR_EL1_NS = X[t];
    else
        ICC_CTLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t];
        else
            ICC_CTLR_EL1_NS = X[t];
    else
        ICC_CTLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_CTLR_EL1_S = X[t];
        else
            ICC_CTLR_EL1_NS = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_DIR_EL1, Interrupt Controller Deactivate Virtual Interrupt Register

The ICV_DIR_EL1 characteristics are:

Purpose

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified virtual interrupt.

Configuration

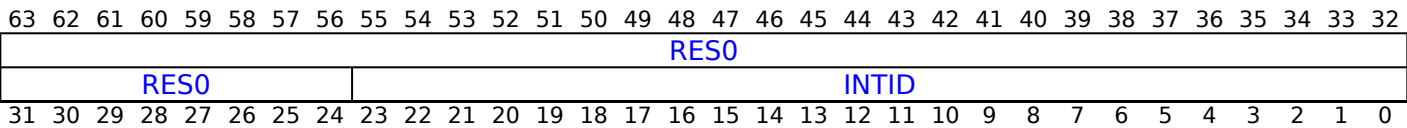
AArch64 System register ICV_DIR_EL1 bits [31:0] performs the same function as AArch32 System register [ICV_DIR\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_DIR_EL1 are UNDEFINED.

Attributes

ICV_DIR_EL1 is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the virtual interrupt to be deactivated.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR_EL1](#).IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_DIR_EL1

When EOImode == 0, writes are ignored. In systems supporting system error generation, an implementation might generate an SEI.

Accesses to this register use the following encodings in the System register encoding space:

MSR ICC_DIR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TDIR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_DIR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_DIR_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_DIR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_DIR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_DIR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_EOIR0_EL1, Interrupt Controller Virtual End Of Interrupt Register 0

The ICV_EOIR0_EL1 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified virtual Group 0 interrupt.

Configuration

AArch64 System register ICV_EOIR0_EL1 performs the same function as AArch32 System register [ICV_EOIR0](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_EOIR0_EL1 are UNDEFINED.

Attributes

ICV_EOIR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding [ICV_IAR0_EL1](#) access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR_EL1](#).IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the [ICV_CTLR](#).EOImode bit is 0, a write to this register drops the priority for the virtual interrupt, and also deactivates the virtual interrupt.

If the [ICV_CTLR](#).EOImode bit is 1, a write to this register only drops the priority for the virtual interrupt. Software must write to [ICV_DIR_EL1](#) to deactivate the virtual interrupt.

Accessing ICV_EOIR0_EL1

A write to this register must correspond to the most recent valid read by this vPE from a Virtual Interrupt Acknowledge Register, and must correspond to the INTID that was read from [ICV_IAR0_EL1](#), otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

Accesses to this register use the following encodings in the System register encoding space:

MSR ICC_EOIR0_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        ICV_EOIR0_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_EOIR0_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                ICC_EOIR0_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_EOIR0_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_EOIR1_EL1, Interrupt Controller Virtual End Of Interrupt Register 1

The ICV_EOIR1_EL1 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified virtual Group 1 interrupt.

Configuration

AArch64 System register ICV_EOIR1_EL1 performs the same function as AArch32 System register [ICV_EOIR1](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_EOIR1_EL1 are UNDEFINED.

Attributes

ICV_EOIR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding [ICV_IAR1_EL1](#) access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR_EL1](#).IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the [ICV_CTLR](#).EOImode bit is 0, a write to this register drops the priority for the virtual interrupt, and also deactivates the virtual interrupt.

If the [ICV_CTLR](#).EOImode bit is 1, a write to this register only drops the priority for the virtual interrupt. Software must write to [ICV_DIR_EL1](#) to deactivate the virtual interrupt.

Accessing ICV_EOIR1_EL1

A write to this register must correspond to the most recent valid read by this vPE from a Virtual Interrupt Acknowledge Register, and must correspond to the INTID that was read from [ICV_IAR1_EL1](#), otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

Accesses to this register use the following encodings in the System register encoding space:

MSR ICC_EOIR1_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        ICV_EOIR1_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_EOIR1_EL1 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_HPPIR0_EL1, Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0

The ICV_HPPIR0_EL1 characteristics are:

Purpose

Indicates the highest priority pending virtual Group 0 interrupt on the virtual CPU interface.

Configuration

AArch64 System register ICV_HPPIR0_EL1 performs the same function as AArch32 System register [ICV_HPPIR0](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_HPPIR0_EL1 are UNDEFINED.

Attributes

ICV_HPPIR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending virtual interrupt.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR_EL1](#).IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_HPPIR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_HPPIR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        return ICV_HPPIR0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR0_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_HPPIR0_EL1;
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR0_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_HPPIR1_EL1, Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1

The ICV_HPPIR1_EL1 characteristics are:

Purpose

Indicates the highest priority pending virtual Group 1 interrupt on the virtual CPU interface.

Configuration

AArch64 System register ICV_HPPIR1_EL1 performs the same function as AArch32 System register [ICV_HPPIR1](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_HPPIR1_EL1 are UNDEFINED.

Attributes

ICV_HPPIR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending virtual interrupt.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR_EL1](#).IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_HPPIR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_HPPIR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        return ICV_HPPIR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR1_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_HPPIR1_EL1;
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_HPPIR1_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_IAR0_EL1, Interrupt Controller Virtual Interrupt Acknowledge Register 0

The ICV_IAR0_EL1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled virtual Group 0 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch64 System register ICV_IAR0_EL1 performs the same function as AArch32 System register [ICV_IAR0](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_IAR0_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when $PSTATE.\{I,F\} == \{0,0\}$). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_IAR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|--|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR_EL1.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_IAR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_IAR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        return ICV_IAR0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IAR0_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IAR0_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_IAR0_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_IAR1_EL1, Interrupt Controller Virtual Interrupt Acknowledge Register 1

The ICV_IAR1_EL1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled virtual Group 1 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch64 System register ICV_IAR1_EL1 performs the same function as AArch32 System register [ICV_IAR1](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_IAR1_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when $PSTATE.\{I,F\} == \{0,0\}$). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_IAR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR_EL1](#).IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_IAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_IAR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        return ICV_IAR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_IAR1_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_IAR1_EL1;
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_IAR1_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_IGRPEN0_EL1, Interrupt Controller Virtual Interrupt Group 0 Enable register

The ICV_IGRPEN0_EL1 characteristics are:

Purpose

Controls whether virtual Group 0 interrupts are enabled or not.

Configuration

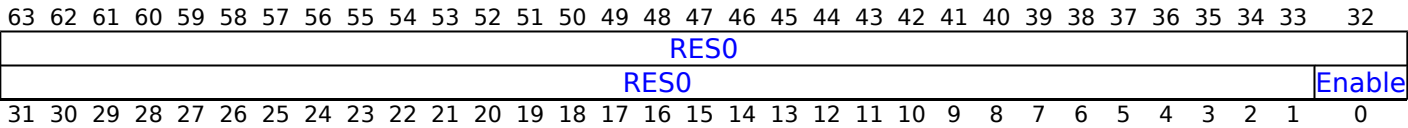
AArch64 System register ICV_IGRPEN0_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ICV_IGRPEN0\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_IGRPEN0_EL1 are UNDEFINED.

Attributes

ICV_IGRPEN0_EL1 is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

Enable, bit [0]

Enables virtual Group 0 interrupts.

| Enable | Meaning |
|--------|--|
| 0b0 | Virtual Group 0 interrupts are disabled. |
| 0b1 | Virtual Group 0 interrupts are enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_IGRPEN0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_IGRPEN0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ICC_IGRPENn_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FMO == '1' then
        return ICV_IGRPEN0_EL1;
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_IGRPEN0_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_IGRPEN0_EL1;
    elsif PSTATE.EL == EL3 then
        if ICC_SRE_EL3.SRE == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_IGRPEN0_EL1;

```

MSR ICC_IGRPEN0_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ICC_IGRPENn_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        ICV_IGRPEN0_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_IGRPEN0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_IGRPEN0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_IGRPEN0_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_IGRPEN1_EL1, Interrupt Controller Virtual Interrupt Group 1 Enable register

The ICV_IGRPEN1_EL1 characteristics are:

Purpose

Controls whether virtual Group 1 interrupts are enabled for the current Security state.

Configuration

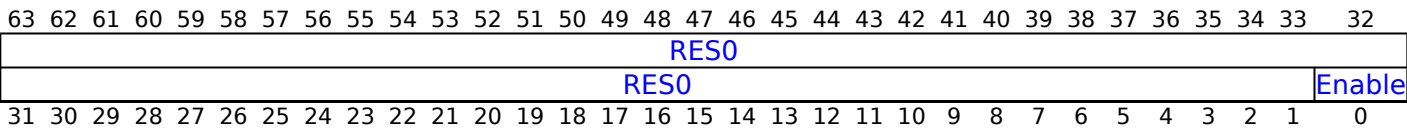
AArch64 System register ICV_IGRPEN1_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ICV_IGRPEN1\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_IGRPEN1_EL1 are UNDEFINED.

Attributes

ICV_IGRPEN1_EL1 is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

Enable, bit [0]

Enables virtual Group 1 interrupts.

| Enable | Meaning |
|--------|--|
| 0b0 | Virtual Group 1 interrupts are disabled. |
| 0b1 | Virtual Group 1 interrupts are enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_IGRPEN1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_IGRPEN1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ICC_IGRPENn_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        return ICV_IGRPEN1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_IGRPEN1_EL1_S;
        else
            return ICC_IGRPEN1_EL1_NS;
    else
        return ICC_IGRPEN1_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_IGRPEN1_EL1_S;
        else
            return ICC_IGRPEN1_EL1_NS;
    else
        return ICC_IGRPEN1_EL1;
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            return ICC_IGRPEN1_EL1_S;
        else
            return ICC_IGRPEN1_EL1_NS;

```

MSR ICC_IGRPEN1_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.ICC_IGRPENn_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        ICV_IGRPEN1_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_IGRPEN1_EL1_S = X[t];
        else
            ICC_IGRPEN1_EL1_NS = X[t];
    else
        ICC_IGRPEN1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_IGRPEN1_EL1_S = X[t];
        else
            ICC_IGRPEN1_EL1_NS = X[t];
    else
        ICC_IGRPEN1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        if SCR_EL3.NS == '0' then
            ICC_IGRPEN1_EL1_S = X[t];
        else
            ICC_IGRPEN1_EL1_NS = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_NMIAR1_EL1, Interrupt Controller Virtual Non-maskable Interrupt Acknowledge Register 1

The ICV_NMIAR1_EL1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled virtual Group 1 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

This register is present only when FEAT_GICv3_NMI is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_NMIAR1_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when $PSTATE.\{I,F\} == \{0,0\}$). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_NMIAR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that virtual interrupt has the Non-maskable property and is of sufficient priority for it to be signalled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR_EL1.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_NMIAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_NMIAR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.IMO == '1' then
        return ICV_NMIAR1_EL1;
    elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return ICC_NMIAR1_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif ICC_SRE_EL2.SRE == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_NMIAR1_EL1;
        elsif PSTATE.EL == EL3 then
            if ICC_SRE_EL3.SRE == '0' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return ICC_NMIAR1_EL1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_PMR_EL1, Interrupt Controller Virtual Interrupt Priority Mask Register

The ICV_PMR_EL1 characteristics are:

Purpose

Provides a virtual interrupt priority filter. Only virtual interrupts with a higher priority than the value in this register are signaled to the PE.

Configuration

AArch64 System register ICV_PMR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ICV_PMR\[31:0\]](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_PMR_EL1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that writes to this register are self-synchronising. This ensures that no interrupts below the written PMR value will be taken after a write to this register is architecturally executed. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_PMR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | Priority | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:8]

Reserved, RES0.

Priority, bits [7:0]

The priority mask level for the virtual CPU interface. If the priority of a virtual interrupt is higher than the value indicated by this field, the interface signals the virtual interrupt to the PE.

The possible priority field values are as follows:

| Implemented priority bits | Possible priority field values | Number of priority levels |
|---------------------------|-------------------------------------|---------------------------|
| [7:0] | 0x00-0xFF (0-255), all values | 256 |
| [7:1] | 0x00-0xFE (0-254), even values only | 128 |
| [7:2] | 0x00-0xFC (0-252), in steps of 4 | 64 |
| [7:3] | 0x00-0xF8 (0-248), in steps of 8 | 32 |
| [7:4] | 0x00-0xF0 (0-240), in steps of 16 | 16 |

Unimplemented priority bits are RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_PMR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_PMR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elseif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.FMO == '1' then
        return ICV_PMR_EL1;
    elseif EL2Enabled() && HCR_EL2.IMO == '1' then
        return ICV_PMR_EL1;
    elseif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_PMR_EL1;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elseif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_PMR_EL1;
elseif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_PMR_EL1;

```

MSR ICC_PMR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FM0 == '1' then
        ICV_PMR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.IM0 == '1' then
        ICV_PMR_EL1 = X[t];
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_PMR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            ICC_PMR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        ICC_PMR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_RPR_EL1, Interrupt Controller Virtual Running Priority Register

The ICV_RPR_EL1 characteristics are:

Purpose

Indicates the Running priority of the virtual CPU interface.

Configuration

AArch64 System register ICV_RPR_EL1 performs the same function as AArch32 System register [ICV_RPR](#).

This register is present only when FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_RPR_EL1 are UNDEFINED.

Attributes

ICV_RPR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| NMI | RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | Priority | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

NMI, bit [63]

When FEAT_GICv3_NMI is implemented:

Indicates whether the running priority is from a NMI.

| NMI | Meaning |
|-----|--|
| 0b0 | There is no active Group 1 NMI, or all active Group 1 NMIs have undergone priority drop. |
| 0b1 | There is an active Group 1 NMI. |

Otherwise:

Reserved, RES0.

Bits [62:8]

Reserved, RES0.

Priority, bits [7:0]

The current running priority on the virtual CPU interface. This is the group priority of the current active virtual interrupt.

If there are no active interrupts on the virtual CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

The priority returned is the group priority as if the BPR for the current Exception level and Security state was set to the minimum value of BPR for the number of implemented priority bits.

Note

If 8 bits of priority are implemented the group priority is bits[7:1] of the priority.

Accessing ICV_RPR_EL1

If there are no active interrupts on the virtual CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

Software cannot determine the number of implemented priority bits from a read of this register.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ICC_RPR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b1011 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elseif ICC_SRE_EL1.SRE == '0' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && ICH_HCR_EL2.TC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.FMO == '1' then
        return ICV_RPR_EL1;
    elseif EL2Enabled() && HCR_EL2.IMO == '1' then
        return ICV_RPR_EL1;
    elseif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_RPR_EL1;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elseif ICC_SRE_EL2.SRE == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_RPR_EL1;
elseif PSTATE.EL == EL3 then
    if ICC_SRE_EL3.SRE == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return ICC_RPR_EL1;

```

ID_AA64AFR0_EL1, AArch64 Auxiliary Feature Register 0

The ID_AA64AFR0_EL1 characteristics are:

Purpose

Provides information about the IMPLEMENTATION DEFINED features of the PE in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

There are no configuration notes.

Attributes

ID_AA64AFR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------------|----|----|----|---------------------------|----|----|----|---------------------------|----|----|----|---------------------------|----|----|----|---------------------------|----|----|----|---------------------------|----|----|----|-----------------|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | IMPLEMENTATION DEFINED | | | | IMPLEMENTATION DEFINED | | | | IMPLEMENTATION DEFINED | | | | IMPLEMENTATION DEFINED | | | | IMPLEMENTATION DEFINED | | | | IMPLEME DEFI | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | | |

Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:28]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [27:24]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [23:20]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [19:16]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [15:12]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [11:8]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [7:4]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [3:0]

IMPLEMENTATION DEFINED.

Accessing ID_AA64AFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64AFR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0101 | 0b100 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64AFR0_EL1;
elseif PSTATE.EL == EL2 then
    return ID_AA64AFR0_EL1;
elseif PSTATE.EL == EL3 then
    return ID_AA64AFR0_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64AFR1_EL1, AArch64 Auxiliary Feature Register 1

The ID_AA64AFR1_EL1 characteristics are:

Purpose

Reserved for future expansion of information about the IMPLEMENTATION DEFINED features of the PE in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

There are no configuration notes.

Attributes

ID_AA64AFR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, RES0.

Accessing ID_AA64AFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64AFR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0101 | 0b101 |

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64AFR1_EL1;
elseif PSTATE.EL == EL2 then
    return ID_AA64AFR1_EL1;
elseif PSTATE.EL == EL3 then
    return ID_AA64AFR1_EL1;
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64DFR0_EL1, AArch64 Debug Feature Register 0

The ID_AA64DFR0_EL1 characteristics are:

Purpose

Provides top level information about the debug system in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

The external register [EDDFR](#) gives information from this register.

Attributes

ID_AA64DFR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------------|----|----|----|----------------------|----|----|----|----------------------|----|----|----|-----------------------|----|----|----|-----------------------------|----|----|----|---------------------------|----|----|----|----------------------------|----|----|----|--------------------------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| HPMN0 | | | | RES0 | | | | BRBE | | | | MTPMU | | | | TraceBuffer | | | | TraceFilt | | | | DoubleLock | | | | PMSVer | | | |
| CTX_CMPs | | | | RES0 | | | | WRPs | | | | RES0 | | | | BRPs | | | | PMUVer | | | | TraceVer | | | | DebugVer | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

HPMN0, bits [63:60]

Zero PMU event counters for a Guest operating system. Defined values are:

| HPMN0 | Meaning |
|--------|--|
| 0b0000 | Setting MDCR_EL2 .HPMN to zero has CONSTRAINED UNPREDICTABLE behavior. |
| 0b0001 | Setting MDCR_EL2 .HPMN to zero has defined behavior. |

All other values are reserved.

If FEAT_PMUv3 is not implemented, FEAT_FGT is not implemented, or EL2 is not implemented, the only permitted value is 0b0000.

FEAT_HPMN0 implements the functionality identified by the value 0b0001.

From Armv8.8, in an implementation that includes FEAT_PMUv3, FEAT_FGT, and EL2, the value 0b0000 is not permitted.

Bits [59:56]

Reserved, RES0.

BRBE, bits [55:52]

Branch Record Buffer Extension. Defined values are:

| BRBE | Meaning |
|--------|--|
| 0b0000 | Branch Record Buffer Extension not implemented. |
| 0b0001 | Branch Record Buffer Extension implemented. |
| 0b0010 | As 0b0001, and adds support for branch recording at EL3. |

All other values are reserved.

FEAT_BRBE implements the functionality identified by the value 0b0001.

FEAT_BRBEv1p1 implements the functionality identified by the value 0b0010.

From Armv9.3, if FEAT_BRBE is implemented, the value 0b0001 is not permitted.

MTPMU, bits [51:48]

Multi-threaded PMU extension. Defined values are:

| MTPMU | Meaning |
|--------|--|
| 0b0000 | FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, it is IMPLEMENTATION DEFINED whether PMEVTYPER<n>_ELO.MT and PMEVTYPER<n>.MT are read/write or RES0. |
| 0b0001 | FEAT_MTPMU and FEAT_PMUv3 implemented. PMEVTYPER<n>_ELO.MT and PMEVTYPER<n>.MT are read/write. When FEAT_MTPMU is disabled, the Effective values of PMEVTYPER<n>_ELO.MT and PMEVTYPER<n>.MT are 0. |
| 0b1111 | FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, PMEVTYPER<n>_ELO.MT and PMEVTYPER<n>.MT are RES0. |

All other values are reserved.

FEAT_MTPMU implements the functionality identified by the value 0b0001.

From Armv8.6, in an implementation that includes FEAT_PMUv3, the value 0b0000 is not permitted.

In an implementation that does not include FEAT_PMUv3, the value 0b0001 is not permitted.

TraceBuffer, bits [47:44]

Trace Buffer Extension. Defined values are:

| TraceBuffer | Meaning |
|-------------|--|
| 0b0000 | Trace Buffer Extension not implemented. |
| 0b0001 | Trace Buffer Extension implemented, FEAT_TRBE. |

All other values are reserved.

TraceFilt, bits [43:40]

Armv8.4 Self-hosted Trace Extension version. Defined values are:

| TraceFilt | Meaning |
|-----------|--|
| 0b0000 | Armv8.4 Self-hosted Trace Extension not implemented. |
| 0b0001 | Armv8.4 Self-hosted Trace Extension implemented. |

All other values are reserved.

FEAT_TRF implements the functionality identified by the value 0b0001.

From Armv8.4, if an Embedded Trace Macrocell Architecture PE Trace Unit is implemented, the value 0b0000 is not permitted.

DoubleLock, bits [39:36]

OS Double Lock implemented. Defined values are:

| DoubleLock | Meaning |
|------------|--|
| 0b0000 | OS Double Lock implemented. OSDLR_EL1 is RW. |
| 0b1111 | OS Double Lock not implemented. OSDLR_EL1 is RAZ/WI. |

All other values are reserved.

FEAT_DoubleLock implements the functionality identified by the value 0b0000.

In Armv8.0, the only permitted value is 0b0000.

If FEAT_Debugv8p2 is implemented and FEAT_DoPD is not implemented, the permitted values are 0b0000 and 0b1111.

If FEAT_DoPD is implemented, the only permitted value is 0b1111.

PMSVer, bits [35:32]

Statistical Profiling Extension version. Defined values are:

| PMSVer | Meaning |
|--------|--|
| 0b0000 | Statistical Profiling Extension not implemented. |
| 0b0001 | Statistical Profiling Extension implemented. |
| 0b0010 | As 0b0001, and adds: <ul style="list-style-type: none"> Support for the Event packet Alignment flag. If FEAT_SVE is implemented, support for the Scalable Vector extensions to Statistical Profiling. |
| 0b0011 | As 0b0010, and adds: <ul style="list-style-type: none"> Discard mode. Extended event filtering, including the PMSNEVER_EL1 System register. Support for the OPTIONAL previous branch target Address packet. If FEAT_PMUv3 is implemented, controls to freeze the PMU event counters after an SPE buffer management event occurs. If FEAT_PMUv3 is implemented, the SAMPLE_FEED_BR, SAMPLE_FEED_EVENT, SAMPLE_FEED_LAT, SAMPLE_FEED_LD, SAMPLE_FEED_OP, and SAMPLE_FEED_ST PMU events. |
| 0b0100 | As 0b0011, and adds: <ul style="list-style-type: none"> If FEAT_MOPS is implemented, Operation Type packet encodings for Memory Copy and Set operations. If FEAT_MTE is implemented, Operation Type packet encodings for loads and stores of Allocation Tags. |

All other values are reserved.

FEAT_SPE implements the functionality identified by the value 0b0001.

FEAT_SPEv1p1 implements the functionality identified by the value 0b0010.

FEAT_SPEv1p2 implements the functionality identified by the value 0b0011.

FEAT_SPEv1p3 implements the functionality identified by the value 0b0100.

From Armv8.5, if FEAT_SPE is implemented, the value 0b0001 is not permitted.

From Armv8.7, if FEAT_SPE is implemented, the value 0b0010 is not permitted.

From Armv8.8, if FEAT_SPE is implemented, the value 0b0011 is not permitted.

CTX_CMPs, bits [31:28]

Number of breakpoints that are context-aware, minus 1. These are the highest numbered breakpoints.

Bits [27:24]

Reserved, RES0.

WRPs, bits [23:20]

Number of watchpoints, minus 1. The value of 0b0000 is reserved.

Bits [19:16]

Reserved, RES0.

BRPs, bits [15:12]

Number of breakpoints, minus 1. The value of 0b0000 is reserved.

PMUVer, bits [11:8]

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version'

Defined values are:

| PMUVer | Meaning |
|--------|--|
| 0b0000 | Performance Monitors Extension not implemented. |
| 0b0001 | Performance Monitors Extension, PMUv3 implemented. |
| 0b0100 | PMUv3 for Armv8.1. As 0b0001, and adds support for: <ul style="list-style-type: none"> Extended 16-bit PMEVTYPER<n>_EL0.evtCount field. If EL2 is implemented, the MDCR_EL2.HPMD control. |
| 0b0101 | PMUv3 for Armv8.4. As 0b0100, and adds support for the PMMIR_EL1 register. |
| 0b0110 | PMUv3 for Armv8.5. As 0b0101, and adds support for: <ul style="list-style-type: none"> 64-bit event counters. If EL2 is implemented, the MDCR_EL2.HCCD control. If EL3 is implemented, the MDCR_EL3.SCCD control. |
| 0b0111 | PMUv3 for Armv8.7. As 0b0110, and adds support for: <ul style="list-style-type: none"> The PMCR_EL0.FZO and, if EL2 is implemented, MDCR_EL2.HPMFZO controls. If EL3 is implemented, the MDCR_EL3.{MPMX,MCCD} controls. |
| 0b1000 | PMUv3 for Armv8.8. As 0b0111, and: <ul style="list-style-type: none"> Extends the Common event number space to include 0x0040 to 0x00BF and 0x4040 to 0x40BF. Removes the CONSTRAINED UNPREDICTABLE behaviors if a reserved or unimplemented PMU event number is selected. |
| 0b1111 | IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value for new implementations. |

All other values are reserved.

FEAT_PMUv3 implements the functionality identified by the value 0b0001.

FEAT_PMUv3p1 implements the functionality identified by the value 0b0100.

FEAT_PMUv3p4 implements the functionality identified by the value 0b0101.

FEAT_PMUv3p5 implements the functionality identified by the value 0b0110.

FEAT_PMUv3p7 implements the functionality identified by the value 0b0111.

FEAT_PMUv3p8 implements the functionality identified by the value 0b1000.

From Armv8.1, if FEAT_PMUv3 is implemented, the value 0b0001 is not permitted.

From Armv8.4, if FEAT_PMUv3 is implemented, the value 0b0100 is not permitted.

From Armv8.5, if FEAT_PMUv3 is implemented, the value 0b0101 is not permitted.

From Armv8.7, if FEAT_PMUv3 is implemented, the value 0b0110 is not permitted.

From Armv8.8, if FEAT_PMUv3 is implemented, the value 0b0111 is not permitted.

TraceVer, bits [7:4]

Trace support. Indicates whether System register interface to a PE trace unit is implemented. Defined values are:

| TraceVer | Meaning |
|----------|---|
| 0b0000 | PE trace unit System registers not implemented. |
| 0b0001 | PE trace unit System registers implemented. |

All other values are reserved.

When PE trace unit System registers are implemented, see [TRCIDR1](#) for tracing capabilities of the trace unit.

DebugVer, bits [3:0]

Debug architecture version. Indicates presence of Armv8 debug architecture. Defined values are:

| DebugVer | Meaning |
|----------|---|
| 0b0110 | Armv8 debug architecture. |
| 0b0111 | Armv8 debug architecture with Virtualization Host Extensions. |
| 0b1000 | Armv8.2 debug architecture, FEAT_Debugv8p2. |
| 0b1001 | Armv8.4 debug architecture, FEAT_Debugv8p4. |
| 0b1010 | Armv8.8 debug architecture, FEAT_Debugv8p8. |

All other values are reserved.

FEAT_VHE adds the functionality identified by the value 0b0111.

FEAT_Debugv8p2 adds the functionality identified by the value 0b1000.

FEAT_Debugv8p4 adds the functionality identified by the value 0b1001.

FEAT_Debugv8p8 adds the functionality identified by the value 0b1010.

From Armv8.1, when FEAT_VHE is implemented the value 0b0110 is not permitted.

From Armv8.2, the values 0b0110 and 0b0111 are not permitted.

From Armv8.4, the value 0b1000 is not permitted.

From Armv8.8, the value 0b1001 is not permitted.

Accessing ID_AA64DFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64DFR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64DFR0_EL1;
elseif PSTATE.EL == EL2 then
    return ID_AA64DFR0_EL1;
elseif PSTATE.EL == EL3 then
    return ID_AA64DFR0_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64DFR1_EL1, AArch64 Debug Feature Register 1

The ID_AA64DFR1_EL1 characteristics are:

Purpose

Reserved for future expansion of top level information about the debug system in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

There are no configuration notes.

Attributes

ID_AA64DFR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, RES0.

Accessing ID_AA64DFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64DFR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64DFR1_EL1;
elseif PSTATE.EL == EL2 then
    return ID_AA64DFR1_EL1;
elseif PSTATE.EL == EL3 then
    return ID_AA64DFR1_EL1;

```


ID_AA64ISAR0_EL1, AArch64 Instruction Set Attribute Register 0

The ID_AA64ISAR0_EL1 characteristics are:

Purpose

Provides information about the instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

There are no configuration notes.

Attributes

ID_AA64ISAR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|----|----|----|----|---------------------|----|----|----|------------------------|----|----|----|-----------------------|----|----|----------------------|----|----|----|----------------------|----|----|---------------------|----|----|----|----------------------|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RNDR | | | | | TLB | | | | TS | | | | FHM | | | DP | | | | SM4 | | | SM3 | | | | SHA3 | | | | |
| RDM | | | | | TME | | | | Atomic | | | | CRC32 | | | SHA2 | | | | SHA1 | | | AES | | | | RES0 | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

RNDR, bits [63:60]

Indicates support for Random Number instructions in AArch64 state.

When FEAT_RNG_TRAP is implemented, the value returned by a direct read of ID_AA64ISAR0_EL1.RNDR is further controlled by the value of [SCR_EL3.TRNDR](#).

Defined values are:

| RNDR | Meaning |
|--------|--|
| 0b0000 | No Random Number instructions are implemented. |
| 0b0001 | RNDR and RNDRRS registers are implemented. |

All other values are reserved.

FEAT_RNG implements the functionality identified by the value 0b0001.

From Armv8.5, the permitted values are 0b0000 and 0b0001.

TLB, bits [59:56]

Indicates support for Outer shareable and TLB range maintenance instructions. Defined values are:

| TLB | Meaning |
|--------|---|
| 0b0000 | Outer shareable and TLB range maintenance instructions are not implemented. |
| 0b0001 | Outer shareable TLB maintenance instructions are implemented. |
| 0b0010 | Outer shareable and TLB range maintenance instructions are implemented. |

All other values are reserved.

FEAT_TLBIOS implements the functionality identified by the values 0b0001 and 0b0010.

FEAT_TLBIRANGE implements the functionality identified by the value 0b0010.

From Armv8.4, the only permitted value is 0b0010.

TS, bits [55:52]

Indicates support for flag manipulation instructions. Defined values are:

| TS | Meaning |
|--------|--|
| 0b0000 | No flag manipulation instructions are implemented. |
| 0b0001 | CFINV, RMIF, SETF16, and SETF8 instructions are implemented. |
| 0b0010 | CFINV, RMIF, SETF16, SETF8, AXFLAG, and XAFLAG instructions are implemented. |

All other values are reserved.

FEAT_FlagM implements the functionality identified by the value 0b0001.

FEAT_FlagM2 implements the functionality identified by the value 0b0010.

In Armv8.2, the permitted values are 0b0000 and 0b0001.

In Armv8.4, the only permitted value is 0b0001.

From Armv8.5, the only permitted value is 0b0010.

FHM, bits [51:48]

Indicates support for FMLAL and FMLSL instructions. Defined values are:

| FHM | Meaning |
|--------|---|
| 0b0000 | FMLAL and FMLSL instructions are not implemented. |
| 0b0001 | FMLAL and FMLSL instructions are implemented. |

All other values are reserved.

FEAT_FHM implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

DP, bits [47:44]

Indicates support for Dot Product instructions in AArch64 state. Defined values are:

| DP | Meaning |
|--------|--|
| 0b0000 | No Dot Product instructions implemented. |
| 0b0001 | UDOT and SDOT instructions implemented. |

All other values are reserved.

FEAT_DotProd implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

SM4, bits [43:40]

Indicates support for SM4 instructions in AArch64 state. Defined values are:

| SM4 | Meaning |
|--------|--|
| 0b0000 | No SM4 instructions implemented. |
| 0b0001 | SM4E and SM4EKEY instructions implemented. |

All other values are reserved.

If FEAT_SM4 is not implemented, the value 0b0001 is reserved.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

This field must have the same value as ID_AA64ISAR0_EL1.SM3.

SM3, bits [39:36]

Indicates support for SM3 instructions in AArch64 state. Defined values are:

| SM3 | Meaning |
|--------|--|
| 0b0000 | No SM3 instructions implemented. |
| 0b0001 | SM3SS1, SM3TT1A, SM3TT1B, SM3TT2A, SM3TT2B, SM3PARTW1, and SM3PARTW2 instructions implemented. |

All other values are reserved.

If FEAT_SM3 is not implemented, the value 0b0001 is reserved.

FEAT_SM3 implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

This field must have the same value as ID_AA64ISAR0_EL1.SM4.

SHA3, bits [35:32]

Indicates support for SHA3 instructions in AArch64 state. Defined values are:

| SHA3 | Meaning |
|--------|---|
| 0b0000 | No SHA3 instructions implemented. |
| 0b0001 | EOR3, RAX1, XAR, and BCAX instructions implemented. |

All other values are reserved.

If FEAT_SHA3 is not implemented, the value 0b0001 is reserved.

FEAT_SHA3 implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR0_EL1.SHA1 is 0b0000, this field must have the value 0b0000.

If the value of this field is 0b0001, ID_AA64ISAR0_EL1.SHA2 must have the value 0b0010.

RDM, bits [31:28]

Indicates support for SQRDMLAH and SQRDMLSH instructions in AArch64 state. Defined values are:

| RDM | Meaning |
|--------|---|
| 0b0000 | No RDMA instructions implemented. |
| 0b0001 | SQRDMLAH and SQRDMLSH instructions implemented. |

All other values are reserved.

FEAT_RDM implements the functionality identified by the value 0b0001.

From Armv8.1, the only permitted value is 0b0001.

TME, bits [27:24]

Indicates support for TME instructions. Defined values are:

| TME | Meaning |
|------------|---|
| 0b0000 | TME instructions are not implemented. |
| 0b0001 | TCANCEL, TCOMMIT, TSTART, and TTEST instructions are implemented. |

If [HCR_EL2](#).TME == 0, reads of this field at EL1 return 0.

If [SCR_EL3](#).TME == 0, reads of this field at EL1 or EL2 return 0.

All other values are reserved.

Atomic, bits [23:20]

Indicates support for Atomic instructions in AArch64 state. Defined values are:

| Atomic | Meaning |
|---------------|--|
| 0b0000 | No Atomic instructions implemented. |
| 0b0010 | LDADD, LDCLR, LDEOR, LDSET, LDSMAX, LDSMIN, LDUMAX, LDUMIN, CAS, CASP, and SWP instructions implemented. |

All other values are reserved.

FEAT_LSE implements the functionality identified by the value 0b0010.

From Armv8.1, the only permitted value is 0b0010.

CRC32, bits [19:16]

Indicates support for CRC32 instructions in AArch64 state. Defined values are:

| CRC32 | Meaning |
|--------------|--|
| 0b0000 | No CRC32 instructions implemented. |
| 0b0001 | CRC32B, CRC32H, CRC32W, CRC32X, CRC32CB, CRC32CH, CRC32CW, and CRC32CX instructions implemented. |

All other values are reserved.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.1, the only permitted value is 0b0001.

SHA2, bits [15:12]

Indicates support for SHA2 instructions in AArch64 state. Defined values are:

| SHA2 | Meaning |
|-------------|---|
| 0b0000 | No SHA2 instructions implemented. |
| 0b0001 | Implements instructions: SHA256H, SHA256H2, SHA256SU0, and SHA256SU1. |
| 0b0010 | Implements instructions: <ul style="list-style-type: none"> SHA256H, SHA256H2, SHA256SU0, and SHA256SU1. SHA512H, SHA512H2, SHA512SU0, and SHA512SU1. |

All other values are reserved.

FEAT_SHA256 implements the functionality identified by the value 0b0001.

FEAT_SHA512 implements the functionality identified by the value 0b0010.

In Armv8, the permitted values are 0b0000 and 0b0001.

From Armv8.2, the permitted values are 0b0000, 0b0001, and 0b0010.

If the value of ID_AA64ISAR0_EL1.SHA1 is 0b0000, this field must have the value 0b0000.

If the value of this field is 0b0010, ID_AA64ISAR0_EL1.SHA3 must have the value 0b0001.

SHA1, bits [11:8]

Indicates support for SHA1 instructions in AArch64 state. Defined values are:

| SHA1 | Meaning |
|--------|--|
| 0b0000 | No SHA1 instructions implemented. |
| 0b0001 | SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, and SHA1SU1 instructions implemented. |

All other values are reserved.

FEAT_SHA1 implements the functionality identified by the value 0b0001.

From Armv8, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR0_EL1.SHA2 is 0b0000, this field must have the value 0b0000.

AES, bits [7:4]

Indicates support for AES instructions in AArch64 state. Defined values are:

| AES | Meaning |
|--------|--|
| 0b0000 | No AES instructions implemented. |
| 0b0001 | AESE, AESD, AESMC, and AESIMC instructions implemented. |
| 0b0010 | As for 0b0001, plus PMULL/PMULL2 instructions operating on 64-bit data quantities. |

FEAT_AES implements the functionality identified by the value 0b0001.

FEAT_PMULL implements the functionality identified by the value 0b0010.

All other values are reserved.

From Armv8, the permitted values are 0b0000 and 0b0010.

Bits [3:0]

Reserved, RES0.

Accessing ID_AA64ISAR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64ISAR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_AA64ISAR0_EL1;
    elsif PSTATE.EL == EL2 then
        return ID_AA64ISAR0_EL1;
    elsif PSTATE.EL == EL3 then
        return ID_AA64ISAR0_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64ISAR1_EL1, AArch64 Instruction Set Attribute Register 1

The ID_AA64ISAR1_EL1 characteristics are:

Purpose

Provides information about the features and instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

There are no configuration notes.

Attributes

ID_AA64ISAR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-----|----|----|----|-------|----|----|----|------|----|----|----|-------|----|----|----|---------|----|----|----|-----|----|----|----|---------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| LS64 | | | | XS | | | | I8MM | | | | DGH | | | | BF16 | | | | SPECRES | | | | SB | | | | FRINTTS | | | |
| GPI | | | | GPA | | | | LRCPC | | | | FCMA | | | | JSCVT | | | | API | | | | APA | | | | DPB | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

LS64, bits [63:60]

Indicates support for LD64B and ST64B* instructions, and the [ACCDATA_EL1](#) register. Defined values of this field are:

| LS64 | Meaning |
|--------|---|
| 0b0000 | The LD64B and ST64B* instructions, the ACCDATA_EL1 register, and associated traps are not supported. |
| 0b0001 | The LD64B and ST64B instructions are supported. |
| 0b0010 | The LD64B, ST64B, and ST64BV instructions, and their associated traps are supported. |
| 0b0011 | The LD64 and ST64B* instructions, the ACCDATA_EL1 register, and their associated traps are supported. |

All other values are reserved.

FEAT_LS64 implements the functionality identified by 0b0001.

FEAT_LS64_V implements the functionality identified by 0b0010.

FEAT_LS64_ACCDATA implements the functionality identified by 0b0011.

From Armv8.7, the permitted values are 0b0000, 0b0001, 0b0010, and 0b0011.

XS, bits [59:56]

Indicates support for the XS attribute, the TLBI and DSB instructions with the nXS qualifier, and the [HCRX_EL2](#).{FGTnXS, FnXS} fields in AArch64 state. Defined values are:

| XS | Meaning |
|--------|--|
| 0b0000 | The XS attribute, the TLBI and DSB instructions with the nXS qualifier, and the HCRX_EL2 .{FGTnXS, FnXS} fields are not supported. |
| 0b0001 | The XS attribute, the TLBI and DSB instructions with the nXS qualifier, and the HCRX_EL2 .{FGTnXS, FnXS} fields are supported. |

All other values are reserved.

FEAT_XS implements the functionality identified by 0b0001.

From Armv8.7, the only permitted value is 0b0001.

I8MM, bits [55:52]

Indicates support for Advanced SIMD and Floating-point Int8 matrix multiplication instructions in AArch64 state. Defined values are:

| I8MM | Meaning |
|--------|--|
| 0b0000 | Int8 matrix multiplication instructions are not implemented. |
| 0b0001 | SMMLA, SUDOT, UMMLA, USMMLA, and USDOT instructions are implemented. |

All other values are reserved.

FEAT_I8MM implements the functionality identified by 0b0001.

When Advanced SIMD and SVE are both implemented, this field must return the same value as [ID_AA64ZFR0_EL1](#).I8MM.

From Armv8.6, the only permitted value is 0b0001.

DGH, bits [51:48]

Indicates support for the Data Gathering Hint instruction. Defined values are:

| DGH | Meaning |
|--------|---|
| 0b0000 | Data Gathering Hint is not implemented. |
| 0b0001 | Data Gathering Hint is implemented. |

All other values are reserved.

FEAT_DGH implements the functionality identified by 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

If the DGH instruction has no effect in preventing the merging of memory accesses, the value of this field is 0b0000.

BF16, bits [47:44]

Indicates support for Advanced SIMD and Floating-point BFloat16 instructions in AArch64 state. Defined values are:

| BF16 | Meaning |
|--------|--|
| 0b0000 | BFloat16 instructions are not implemented. |
| 0b0001 | BFCVT, BFCVTN, BFCVTN2, BFDOT, BFMLALB, BFMLALT, and BFMMMLA instructions are implemented. |
| 0b0010 | As 0b0001, but the FPCR .EBF field is also supported. |

All other values are reserved.

FEAT_BF16 implements the functionality identified by 0b0001.

FEAT_EBF16 implements the functionality identified by 0b0010.

When Advanced SIMD and SVE are both implemented, this field must return the same value as [ID_AA64ZFR0_EL1](#).BF16.

From Armv8.6, if FEAT_SME is implemented, the permitted values are 0b0001 and 0b0010.

From Armv8.6, if FEAT_SME is not implemented, the only permitted value is 0b0001.

SPECRES, bits [43:40]

Indicates support for prediction invalidation instructions in AArch64 state. Defined values are:

| SPECRES | Meaning |
|---------|--|
| 0b0000 | CFP RCTX, DVP RCTX, and CPP RCTX instructions are not implemented. |
| 0b0001 | CFP RCTX, DVP RCTX, and CPP RCTX instructions are implemented. |

All other values are reserved.

FEAT_SPECRES implements the functionality identified by 0b0001.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

SB, bits [39:36]

Indicates support for SB instruction in AArch64 state. Defined values are:

| SB | Meaning |
|--------|------------------------------------|
| 0b0000 | SB instruction is not implemented. |
| 0b0001 | SB instruction is implemented. |

All other values are reserved.

FEAT_SB implements the functionality identified by 0b0001.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

FRINTTS, bits [35:32]

Indicates support for the FRINT32Z, FRINT32X, FRINT64Z, and FRINT64X instructions are implemented. Defined values are:

| FRINTTS | Meaning |
|---------|--|
| 0b0000 | FRINT32Z, FRINT32X, FRINT64Z, and FRINT64X instructions are not implemented. |
| 0b0001 | FRINT32Z, FRINT32X, FRINT64Z, and FRINT64X instructions are implemented. |

All other values are reserved.

FEAT_FRINTTS implements the functionality identified by 0b0001.

From Armv8.5, the only permitted value is 0b0001.

GPI, bits [31:28]

Indicates support for an IMPLEMENTATION DEFINED algorithm is implemented in the PE for generic code authentication in AArch64 state. Defined values are:

| GPI | Meaning |
|--------|---|
| 0b0000 | Generic Authentication using an IMPLEMENTATION DEFINED algorithm is not implemented. |
| 0b0001 | Generic Authentication using an IMPLEMENTATION DEFINED algorithm is implemented. This includes the PACGA instruction. |

All other values are reserved.

FEAT_PACIMP implements the functionality identified by 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.GPA is non-zero, or the value of [ID_AA64ISAR2_EL1.GPA3](#) is non-zero, this field must have the value 0b0000.

GPA, bits [27:24]

Indicates whether the QARMA5 algorithm is implemented in the PE for generic code authentication in AArch64 state. Defined values are:

| GPA | Meaning |
|--------|--|
| 0b0000 | Generic Authentication using the QARMA5 algorithm is not implemented. |
| 0b0001 | Generic Authentication using the QARMA5 algorithm is implemented. This includes the PACGA instruction. |

All other values are reserved.

FEAT_PACQARMA5 implements the functionality identified by 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of ID_AA64ISAR1_EL1.GPI is non-zero, or the value of [ID_AA64ISAR2_EL1.GPA3](#) is non-zero, this field must have the value 0b0000.

LRCPC, bits [23:20]

Indicates support for weaker release consistency, RCpc, based model. Defined values are:

| LRCPC | Meaning |
|--------|---|
| 0b0000 | The LDAPR*, LDAPUR*, and STLUR* instructions are not implemented. |
| 0b0001 | The LDAPR* instructions are implemented. The LDAPUR*, and STLUR* instructions are not implemented. |
| 0b0010 | The LDAPR*, LDAPUR*, and STLUR* instructions are implemented. |

All other values are reserved.

FEAT_LRCPC implements the functionality identified by the value 0b0001.

FEAT_LRCPC2 implements the functionality identified by the value 0b0010.

In Armv8.2, the permitted values are 0b0000, 0b0001, and 0b0010.

In Armv8.3, the permitted values are 0b0001 and 0b0010.

From Armv8.4, the only permitted value is 0b0010.

FCMA, bits [19:16]

Indicates support for complex number addition and multiplication, where numbers are stored in vectors. Defined values are:

| FCMA | Meaning |
|--------|---|
| 0b0000 | The FCMLA and FCADD instructions are not implemented. |
| 0b0001 | The FCMLA and FCADD instructions are implemented. |

All other values are reserved.

FEAT_FCMA implements the functionality identified by the value 0b0001.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

From Armv8.3, if Advanced SIMD or Floating-point is implemented, the only permitted value is 0b0001.

From Armv8.3, if Advanced SIMD or Floating-point is not implemented, the only permitted value is 0b0000.

JSCVT, bits [15:12]

Indicates support for JavaScript conversion from double precision floating point values to integers in AArch64 state. Defined values are:

| JSCVT | Meaning |
|--------|---|
| 0b0000 | The FJCVTZS instruction is not implemented. |
| 0b0001 | The FJCVTZS instruction is implemented. |

All other values are reserved.

FEAT_JSCVT implements the functionality identified by 0b0001.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

From Armv8.3, if Advanced SIMD or Floating-point is implemented, the only permitted value is 0b0001.

From Armv8.3, if Advanced SIMD or Floating-point is not implemented, the only permitted value is 0b0000.

API, bits [11:8]

Indicates whether an IMPLEMENTATION DEFINED algorithm is implemented in the PE for address authentication, in AArch64 state. This applies to all Pointer Authentication instructions other than the PACGA instruction. Defined values are:

| API | Meaning |
|--------|--|
| 0b0000 | Address Authentication using an IMPLEMENTATION DEFINED algorithm is not implemented. |
| 0b0001 | Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC() and HaveEnhancedPAC2() functions returning FALSE. |
| 0b0010 | Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC() function returning TRUE, and the HaveEnhancedPAC2() function returning FALSE. |
| 0b0011 | Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, and the HaveEnhancedPAC() function returning FALSE. |
| 0b0100 | Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning FALSE, and the HaveEnhancedPAC() function returning FALSE. |
| 0b0101 | Address Authentication using an IMPLEMENTATION DEFINED algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning TRUE, and the HaveEnhancedPAC() function returning FALSE. |

All other values are reserved.

FEAT_PAuth implements the functionality identified by 0b0001.

FEAT_EPAC implements the functionality identified by 0b0010.

FEAT_PAuth2 implements the functionality identified by 0b0011.

FEAT_FPAC implements the functionality identified by 0b0100.

FEAT_FPACCOMBINE implements the functionality identified by 0b0101.

When this field is non-zero, FEAT_PACIMP is implemented.

In Armv8.3, the permitted values are 0b0001, 0b0010, 0b0011, 0b0100, and 0b0101.

From Armv8.6, the permitted values are 0b0011, 0b0100, and 0b0101.

If the value of ID_AA64ISAR1_EL1.APA is non-zero, or the value of [ID_AA64ISAR2_EL1.APA3](#) is non-zero, this field must have the value 0b0000.

APA, bits [7:4]

Indicates whether the QARMA5 algorithm is implemented in the PE for address authentication, in AArch64 state. This applies to all Pointer Authentication instructions other than the PACGA instruction. Defined values are:

| APA | Meaning |
|--------|--|
| 0b0000 | Address Authentication using the QARMA5 algorithm is not implemented. |
| 0b0001 | Address Authentication using the QARMA5 algorithm is implemented, with the HaveEnhancedPAC() and HaveEnhancedPAC2() functions returning FALSE. |
| 0b0010 | Address Authentication using the QARMA5 algorithm is implemented, with the HaveEnhancedPAC() function returning TRUE and the HaveEnhancedPAC2() function returning FALSE. |
| 0b0011 | Address Authentication using the QARMA5 algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning FALSE, the HaveFPACCombined() function returning FALSE, and the HaveEnhancedPAC() function returning FALSE. |
| 0b0100 | Address Authentication using the QARMA5 algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning FALSE, and the HaveEnhancedPAC() function returning FALSE. |
| 0b0101 | Address Authentication using the QARMA5 algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning TRUE, and the HaveEnhancedPAC() function returning FALSE. |

All other values are reserved.

FEAT_PAuth implements the functionality identified by 0b0001.

FEAT_EPAC implements the functionality identified by 0b0010.

FEAT_PAuth2 implements the functionality identified by 0b0011.

FEAT_FPAC implements the functionality identified by 0b0100.

FEAT_FPACCOMBINE implements the functionality identified by 0b0101.

When this field is non-zero, FEAT_PACQARMA5 is implemented.

In Armv8.3, the permitted values are 0b0001, 0b0010, 0b0011, 0b0100, and 0b0101.

From Armv8.6, the permitted values are 0b0011, 0b0100, and 0b0101.

If the value of ID_AA64ISAR1_EL1.API is non-zero, or the value of [ID_AA64ISAR2_EL1.APA3](#) is non-zero, this field must have the value 0b0000.

DPB, bits [3:0]

Data Persistence writeback. Indicates support for the [DC CVAP](#) and [DC CVADP](#) instructions in AArch64 state. Defined values are:

| DPB | Meaning |
|--------|---|
| 0b0000 | DC CVAP not supported. |
| 0b0001 | DC CVAP supported. |
| 0b0010 | DC CVAP and DC CVADP supported. |

All other values are reserved.

FEAT_DPB implements the functionality identified by the value 0b0001.

FEAT_DPB2 implements the functionality identified by the value 0b0010.

In Armv8.2, the permitted values are 0b0001 and 0b0010.

From Armv8.5, the only permitted value is 0b0010.

Accessing ID_AA64ISAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64ISAR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0110 | 0b001 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_AA64ISAR1_EL1;
    elsif PSTATE.EL == EL2 then
        return ID_AA64ISAR1_EL1;
    elsif PSTATE.EL == EL3 then
        return ID_AA64ISAR1_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64ISAR2_EL1, AArch64 Instruction Set Attribute Register 2

The ID_AA64ISAR2_EL1 characteristics are:

Purpose

Provides information about the features and instructions implemented in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64ISAR2_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----------|----|----|----|----|----|----|----|------|----|----|----|------|----|----|----|------|----|----|----|-------|----|----|----|------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | PAC_frac | | | | BC | | | | MOPS | | | | APA3 | | | | GPA3 | | | | RPRES | | | | WFXt | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:28]

Reserved, RES0.

PAC_frac, bits [27:24]

Indicates whether the ConstPACField() function used as part of the PAC addition returns FALSE or TRUE.

| PAC_frac | Meaning |
|----------|--------------------------------|
| 0b0000 | ConstPACField() returns FALSE. |
| 0b0001 | ConstPACField() returns TRUE. |

All other values are reserved.

FEAT_CONSTPACFIELD implements the functionality identified by 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

BC, bits [23:20]

Indicates support for the BC instruction in AArch64 state. Defined values are:

| BC | Meaning |
|--------|------------------------------------|
| 0b0000 | BC instruction is not implemented. |
| 0b0001 | BC instruction is implemented. |

All other values are reserved.

FEAT_HBC implements the functionality identified by the value 0b0001.

From Armv8.8, the only permitted value is 0b0001.

MOPS, bits [19:16]

Indicates support for the Memory Copy and Memory Set instructions in AArch64 state.

| MOPS | Meaning |
|--------|--|
| 0b0000 | The Memory Copy and Memory Set instructions are not implemented in AArch64 state. |
| 0b0001 | The Memory Copy and Memory Set instructions are implemented in AArch64 state with the following exception. If FEAT_MTE is implemented, then SETGP*, SETGM* and SETGE* instructions are also supported. |

All other values are reserved.

FEAT_MOPS implements the functionality identified by the value 0b0001.

From Armv8.8, the only permitted value is 0b0001.

APA3, bits [15:12]

Indicates whether the QARMA3 algorithm is implemented in the PE for address authentication in AArch64 state. This applies to all Pointer Authentication instructions other than the PACGA instruction. Defined values are:

| APA3 | Meaning |
|--------|--|
| 0b0000 | Address Authentication using the QARMA3 algorithm is not implemented. |
| 0b0001 | Address Authentication using the QARMA3 algorithm is implemented, with the HaveEnhancedPAC() and HaveEnhancedPAC2() functions returning FALSE. |
| 0b0010 | Address Authentication using the QARMA3 algorithm is implemented, with the HaveEnhancedPAC() function returning TRUE and the HaveEnhancedPAC2() function returning FALSE. |
| 0b0011 | Address Authentication using the QARMA3 algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning FALSE, the HaveFPACCombined() function returning FALSE, and the HaveEnhancedPAC() function returning FALSE. |
| 0b0100 | Address Authentication using the QARMA3 algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning FALSE, and the HaveEnhancedPAC() function returning FALSE. |
| 0b0101 | Address Authentication using the QARMA3 algorithm is implemented, with the HaveEnhancedPAC2() function returning TRUE, the HaveFPAC() function returning TRUE, the HaveFPACCombined() function returning TRUE, and the HaveEnhancedPAC() function returning FALSE. |

All other values are reserved.

FEAT_PAuth implements the functionality identified by 0b0001.

FEAT_EPAC implements the functionality identified by 0b0010.

FEAT_PAuth2 implements the functionality identified by 0b0011.

FEAT_FPAC implements the functionality identified by 0b0100.

FEAT_FPACCOMBINE implements the functionality identified by 0b0101.

When this field is non-zero, FEAT_PACQARMA3 is implemented.

In Armv8.3, the permitted values are 0b0000, 0b0001, 0b0010, 0b0011, 0b0100, and 0b0101.

From Armv8.6, the permitted values are 0b0011, 0b0100, and 0b0101.

If the value of [ID_AA64ISAR1_EL1.API](#) is non-zero, or the value of [ID_AA64ISAR1_EL1.APA](#) is non-zero, this field must have the value 0b0000.

GPA3, bits [11:8]

Indicates whether the QARMA3 algorithm is implemented in the PE for generic code authentication in AArch64 state. Defined values are:

| GPA3 | Meaning |
|--------|--|
| 0b0000 | Generic Authentication using the QARMA3 algorithm is not implemented. |
| 0b0001 | Generic Authentication using the QARMA3 algorithm is implemented. This includes the PACGA instruction. |

All other values are reserved.

FEAT_PACQARMA3 implements the functionality identified by 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

If the value of [ID_AA64ISAR1_EL1.GPI](#) is non-zero, or the value of [ID_AA64ISAR1_EL1.GPA](#) is non-zero, this field must have the value 0b0000.

RPRES, bits [7:4]

When [FPCR.AH](#) is 1, indicates support for 12 bits of mantissa in reciprocal and reciprocal square root instructions in AArch64 state. Defined values are:

| RPRES | Meaning |
|--------|---|
| 0b0000 | Reciprocal and reciprocal square root estimates give 8 bits of mantissa. |
| 0b0001 | Reciprocal and reciprocal square root estimates give 12 bits of mantissa. |

All other values are reserved.

FEAT_RPRES implements the functionality identified by the value 0b0001.

From Armv8.7, if Advanced SIMD and floating-point is implemented, the only permitted value is 0b0001.

WFxT, bits [3:0]

Indicates support for the WFET and WFIT instructions in AArch64 state. Defined values are:

| WFxT | Meaning |
|--------|--|
| 0b0000 | WFET and WFIT are not supported. |
| 0b0010 | WFET and WFIT are supported, and the register number is reported in the ESR_ELx on exceptions. |

All other values are reserved.

FEAT_WFxT implements the functionality identified by the value 0b0010.

From Armv8.7, the only permitted value is 0b0010.

Accessing ID_AA64ISAR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64ISAR2_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0110 | 0b010 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_AA64ISAR2_EL1;
    elsif PSTATE.EL == EL2 then
        return ID_AA64ISAR2_EL1;
    elsif PSTATE.EL == EL3 then
        return ID_AA64ISAR2_EL1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64MMFR0_EL1, AArch64 Memory Model Feature Register 0

The ID_AA64MMFR0_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

There are no configuration notes.

Attributes

ID_AA64MMFR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|---------|----|----|----|---------|----|----|----|-----------|----|----|----|--------|----|----|----|----------|----|----|----|-----------|----|----|----|-----------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ECV | | | | FGT | | | | RES0 | | | | | | | | ExS | | | | TGran4_2 | | | | TGran64_2 | | | | TGran16_2 | | | |
| TGran4 | | | | TGran64 | | | | TGran16 | | | | BigEndEL0 | | | | SNSMem | | | | BigEnd | | | | ASIDBits | | | | PARange | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ECV, bits [63:60]

Indicates presence of Enhanced Counter Virtualization. Defined values are:

| ECV | Meaning |
|--------|--|
| 0b0000 | Enhanced Counter Virtualization is not implemented. |
| 0b0001 | Enhanced Counter Virtualization is implemented. Supports CNTHCTL_EL2 .{EL1TVT, EL1TVCT, EL1NVPCT, EL1NVVCT, EVNTIS}, CNTKCTL_EL1 .EVNTIS, CNTPCTSS_EL0 counter views, and CNTVCTSS_EL0 counter views. Extends the PMSCR_EL1 .PCT, PMSCR_EL2 .PCT, TRFCR_EL1 .TS, and TRFCR_EL2 .TS fields. |
| 0b0010 | As 0b0001, and also includes support for CNTHCTL_EL2 .ECV and CNTPOFF_EL2 . |

All other values are reserved.

FEAT_ECV implements the functionality identified by the values 0b0001 and 0b0010.

From Armv8.6, the only permitted values are 0b0001 and 0b0010.

FGT, bits [59:56]

Indicates presence of the Fine-Grained Trap controls:

- If EL2 is implemented, the [HAFGRTR_EL2](#), [HDFGRTR_EL2](#), [HDFGWTR_EL2](#), [HFGRTR_EL2](#), [HFGITR_EL2](#) and [HFGWTR_EL2](#) registers, and their associated traps.
- If EL2 is implemented, [MDCR_EL2](#).TDCC.
- If EL3 is implemented, [MDCR_EL3](#).TDCC.
- If both EL2 and EL3 are implemented, [SCR_EL3](#).FGTEn.

Defined values are:

| FGT | Meaning |
|--------|---|
| 0b0000 | The fine-grained trap controls are not implemented. |
| 0b0001 | The fine-grained trap controls are implemented. |

All other values are reserved.

FEAT_FGT implements the functionality identified by the value 0b0001.

From Armv8.6, the only permitted value is 0b0001.

Bits [55:48]

Reserved, RES0.

ExS, bits [47:44]

Indicates support for disabling context synchronizing exception entry and exit. Defined values are:

| ExS | Meaning |
|--------|---|
| 0b0000 | All exception entries and exits are context synchronization events. |
| 0b0001 | Non-context synchronizing exception entry and exit are supported. |

All other values are reserved.

FEAT_ExS implements the functionality identified by the value 0b0001.

TGran4_2, bits [43:40]

Indicates support for 4KB memory granule size at stage 2. Defined values are:

| TGran4_2 | Meaning | Applies when |
|----------|--|-------------------------------|
| 0b0000 | Support for 4KB granule at stage 2 is identified in the ID_AA64MMFR0_EL1.TGran4 field. | When FEAT_LPA2 is implemented |
| 0b0001 | 4KB granule not supported at stage 2. | |
| 0b0010 | 4KB granule supported at stage 2. | |
| 0b0011 | 4KB granule at stage 2 supports 52-bit input and output addresses. | |

All other values are reserved.

The 0b0000 value is deprecated.

Note

This field does not follow the standard ID scheme. See Alternative ID scheme used for ID_AA64MMFR0_EL1 stage 2 granule sizes for more information.

TGran64_2, bits [39:36]

Indicates support for 64KB memory granule size at stage 2. Defined values are:

| TGran64_2 | Meaning |
|-----------|--|
| 0b0000 | Support for 64KB granule at stage 2 is identified in the ID_AA64MMFR0_EL1.TGran64 field. |
| 0b0001 | 64KB granule not supported at stage 2. |
| 0b0010 | 64KB granule supported at stage 2. |

All other values are reserved.

The 0b0000 value is deprecated.

Note

This field does not follow the standard ID scheme. See Alternative ID scheme used for ID_AA64MMFR0_EL1 stage 2 granule sizes for more information.

TGran16_2, bits [35:32]

Indicates support for 16KB memory granule size at stage 2. Defined values are:

| TGran16_2 | Meaning | Applies when |
|------------------|--|-------------------------------|
| 0b0000 | Support for 16KB granule at stage 2 is identified in the ID_AA64MMFR0_EL1.TGran16 field. | When FEAT_LPA2 is implemented |
| 0b0001 | 16KB granule not supported at stage 2. | |
| 0b0010 | 16KB granule supported at stage 2. | |
| 0b0011 | 16KB granule at stage 2 supports 52-bit input and output addresses. | |

All other values are reserved.

The 0b0000 value is deprecated.

Note

This field does not follow the standard ID scheme. See Alternative ID scheme used for ID_AA64MMFR0_EL1 stage 2 granule sizes for more information.

TGran4, bits [31:28]

Indicates support for 4KB memory translation granule size. Defined values are:

| TGran4 | Meaning | Applies when |
|---------------|---|-------------------------------|
| 0b0000 | 4KB granule supported. | When FEAT_LPA2 is implemented |
| 0b0001 | 4KB granule supports 52-bit input and output addresses. | |
| 0b1111 | 4KB granule not supported. | |

All other values are reserved.

TGran64, bits [27:24]

Indicates support for 64KB memory translation granule size. Defined values are:

| TGran64 | Meaning |
|----------------|-----------------------------|
| 0b0000 | 64KB granule supported. |
| 0b1111 | 64KB granule not supported. |

All other values are reserved.

TGran16, bits [23:20]

Indicates support for 16KB memory translation granule size. Defined values are:

| TGran16 | Meaning | Applies when |
|---------|--|-------------------------------|
| 0b0000 | 16KB granule not supported. | When FEAT_LPA2 is implemented |
| 0b0001 | 16KB granule supported. | |
| 0b0010 | 16KB granule supports 52-bit input and output addresses. | |

All other values are reserved.

BigEndEL0, bits [19:16]

Indicates support for mixed-endian at EL0 only. Defined values are:

| BigEndEL0 | Meaning |
|-----------|--|
| 0b0000 | No mixed-endian support at EL0. The SCTLR_EL1.E0E bit has a fixed value. |
| 0b0001 | Mixed-endian support at EL0. The SCTLR_EL1.E0E bit can be configured. |

All other values are reserved.

This field is invalid and is RES0 if ID_AA64MMFR0_EL1.BigEnd is not 0b0000.

SNSMem, bits [15:12]

Indicates support for a distinction between Secure and Non-secure Memory. Defined values are:

| SNSMem | Meaning |
|--------|--|
| 0b0000 | Does not support a distinction between Secure and Non-secure Memory. |
| 0b0001 | Does support a distinction between Secure and Non-secure Memory. |

Note

If EL3 is implemented, the value 0b0000 is not permitted.

All other values are reserved.

BigEnd, bits [11:8]

Indicates support for mixed-endian configuration. Defined values are:

| BigEnd | Meaning |
|--------|---|
| 0b0000 | No mixed-endian support. The SCTLR_ELx.EE bits have a fixed value. See the BigEndEL0 field, bits[19:16], for whether EL0 supports mixed-endian. |
| 0b0001 | Mixed-endian support. The SCTLR_ELx.EE and SCTLR_EL1.E0E bits can be configured. |

All other values are reserved.

ASIDBits, bits [7:4]

Number of ASID bits. Defined values are:

| ASIDBits | Meaning |
|----------|----------|
| 0b0000 | 8 bits. |
| 0b0010 | 16 bits. |

All other values are reserved.

PARange, bits [3:0]

Physical Address range supported. Defined values are:

| PARange | Meaning |
|---------|-----------------|
| 0b0000 | 32 bits, 4GB. |
| 0b0001 | 36 bits, 64GB. |
| 0b0010 | 40 bits, 1TB. |
| 0b0011 | 42 bits, 4TB. |
| 0b0100 | 44 bits, 16TB. |
| 0b0101 | 48 bits, 256TB. |
| 0b0110 | 52 bits, 4PB. |

All other values are reserved.

The value 0b0110 is permitted only if the implementation includes FEAT_LPA, otherwise it is reserved.

Accessing ID_AA64MMFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64MMFR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0111 | 0b000 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64MMFR0_EL1;
elseif PSTATE.EL == EL2 then
    return ID_AA64MMFR0_EL1;
elseif PSTATE.EL == EL3 then
    return ID_AA64MMFR0_EL1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64MMFR1_EL1, AArch64 Memory Model Feature Register 1

The ID_AA64MMFR1_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

There are no configuration notes.

Attributes

ID_AA64MMFR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|---------|----|----|----|--------|----|----|----|--------|----|----|----|------|----|----|----|-----|----|----|----------|----|----|----|----|--------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | CMOW | | | | TIDCP1 | | | | nTLBPA | | | | AFP | | | | HCX | | | ETS | | | | | TWED | | | |
| XNX | | | | SpecSEI | | | | PAN | | | | LO | | | | HPDS | | | | VH | | | VMIDBits | | | | | HAFDBS | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:60]

Reserved, RES0.

CMOW, bits [59:56]

Indicates support for cache maintenance instruction permission. Defined values are:

| CMOW | Meaning |
|--------|---|
| 0b0000 | SCTLR_EL1 .CMOW, SCTLR_EL2 .CMOW, and HCRX_EL2 .CMOW bits are not implemented. |
| 0b0001 | SCTLR_EL1 .CMOW is implemented. If EL2 is implemented, SCTLR_EL2 .CMOW and HCRX_EL2 .CMOW bits are implemented. |

All other values are reserved.

FEAT_CMOW implements the functionality identified by the value 0b0001.

From Armv8.8, the only permitted value is 0b0001.

TIDCP1, bits [55:52]

Indicates whether [SCTLR_EL1](#).TIDCP and [SCTLR_EL2](#).TIDCP are implemented in AArch64 state. Defined values are:

| TIDCP1 | Meaning |
|---------------|--|
| 0b0000 | SCTLR_EL1 .TIDCP and SCTLR_EL2 .TIDCP bits are not implemented and are RES0. |
| 0b0001 | SCTLR_EL1 .TIDCP bit is implemented. If EL2 is implemented, SCTLR_EL2 .TIDCP bit is implemented. |

All other values are reserved.

FEAT_TIDCP1 implements the functionality identified by the value 0b0001.

From Armv8.8, the only permitted value is 0b0001.

nTLBPA, bits [51:48]

Indicates support for intermediate caching of translation table walks. Defined values are:

| nTLBPA | Meaning |
|---------------|--|
| 0b0000 | The intermediate caching of translation table walks might include non-coherent caches of previous valid translation table entries since the last completed relevant TLBI applicable to the PE where either: <ul style="list-style-type: none"> The caching is indexed by the physical address of the location holding the translation table entry. The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry. |
| 0b0001 | The intermediate caching of translation table walks does not include non-coherent caches of previous valid translation table entries since the last completed TLBI applicable to the PE where either: <ul style="list-style-type: none"> The caching is indexed by the physical address of the location holding the translation table entry. The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry. |

All other values are reserved.

FEAT_nTLBPA implements the functionality identified by the value 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

AFP, bits [47:44]

Indicates support for [FPCR](#).{AH, FIZ, NEP}. Defined values are:

| AFP | Meaning |
|------------|--|
| 0b0000 | The FPCR .{AH, FIZ, NEP} fields are not supported. |
| 0b0001 | The FPCR .{AH, FIZ, NEP} fields are supported. |

All other values are reserved.

FEAT_AFP implements the functionality identified by the value 0b0001.

From Armv8.7, if Advanced SIMD and floating-point is implemented, the only permitted value is 0b0001.

HCX, bits [43:40]

Indicates support for [HCRX_EL2](#) and its associated EL3 trap. Defined values are:

| HCX | Meaning |
|------------|---|
| 0b0000 | HCRX_EL2 and its associated EL3 trap are not supported. |
| 0b0001 | HCRX_EL2 and its associated EL3 trap are supported. |

All other values are reserved.

FEAT_HCX implements the functionality identified by the value 0b0001.

From Armv8.7, if EL2 is implemented, the only permitted value is 0b0001.

ETS, bits [39:36]

Indicates support for Enhanced Translation Synchronization. Defined values are:

| ETS | Meaning |
|--------|--|
| 0b0000 | Enhanced Translation Synchronization is not supported. |
| 0b0001 | Enhanced Translation Synchronization is supported. |

All other values are reserved.

FEAT_ETTS implements the functionality identified by the value 0b0001.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.7, the only permitted value is 0b0001.

TWED, bits [35:32]

Indicates support for the configurable delayed trapping of WFE. Defined values are:

| TWED | Meaning |
|--------|--|
| 0b0000 | Configurable delayed trapping of WFE is not supported. |
| 0b0001 | Configurable delayed trapping of WFE is supported. |

All other values are reserved.

FEAT_TWED implements the functionality identified by the value 0b0001.

From Armv8.6, the permitted values are 0b0000 and 0b0001.

XNX, bits [31:28]

Indicates support for execute-never control distinction by Exception level at stage 2. Defined values are:

| XNX | Meaning |
|--------|---|
| 0b0000 | Distinction between EL0 and EL1 execute-never control at stage 2 not supported. |
| 0b0001 | Distinction between EL0 and EL1 execute-never control at stage 2 supported. |

All other values are reserved.

FEAT_XNX implements the functionality identified by the value 0b0001.

From Armv8.2, the only permitted value is 0b0001.

SpecSEI, bits [27:24]

Describes whether the PE can generate SError interrupt exceptions from speculative reads of memory, including speculative instruction fetches. The defined values of this field are:

| SpecSEI | Meaning |
|---------|--|
| 0b0000 | The PE never generates an SError interrupt due to an External abort on a speculative read. |
| 0b0001 | The PE might generate an SError interrupt due to an External abort on a speculative read. |

All other values are reserved.

PAN, bits [23:20]

Privileged Access Never. Indicates support for the PAN bit in PSTATE, [SPSR_EL1](#), [SPSR_EL2](#), [SPSR_EL3](#), and [DPSR_EL0](#). Defined values are:

| PAN | Meaning |
|--------|--|
| 0b0000 | PAN not supported. |
| 0b0001 | PAN supported. |
| 0b0010 | PAN supported and AT S1E1RP and AT S1E1WP instructions supported. |
| 0b0011 | PAN supported, AT S1E1RP and AT S1E1WP instructions supported, and SCTLR_EL1.EPAN and SCTLR_EL2.EPAN bits supported. |

All other values are reserved.

FEAT_PAN implements the functionality identified by the value 0b0001.

FEAT_PAN2 implements the functionality added by the value 0b0010.

FEAT_PAN3 implements the functionality added by the value 0b0011.

In Armv8.1, the permitted values are 0b0001, 0b0010, and 0b0011.

From Armv8.2, the permitted values are 0b0010 and 0b0011.

From Armv8.7, the only permitted value is 0b0011.

LO, bits [19:16]

LORegions. Indicates support for LORegions. Defined values are:

| LO | Meaning |
|--------|--------------------------|
| 0b0000 | LORegions not supported. |
| 0b0001 | LORegions supported. |

All other values are reserved.

FEAT_LOR implements the functionality identified by the value 0b0001.

From Armv8.1, the only permitted value is 0b0001.

HPDS, bits [15:12]

Hierarchical Permission Disables. Indicates support for disabling hierarchical controls in translation tables. Defined values are:

| HPDS | Meaning |
|--------|--|
| 0b0000 | Disabling of hierarchical controls not supported. |
| 0b0001 | Disabling of hierarchical controls supported with the TCR_EL1 .{HPD1, HPD0}, TCR_EL2 .HPD or TCR_EL2 .{HPD1, HPD0}, and TCR_EL3 .HPD bits. |
| 0b0010 | As for value 0b0001, and adds possible hardware allocation of bits[62:59] of the translation table descriptors from the final lookup level for IMPLEMENTATION DEFINED use. |

All other values are reserved.

FEAT_HPDS implements the functionality identified by the value 0b0001.

FEAT_HPDS2 implements the functionality identified by the value 0b0010.

From Armv8.1, the value 0b0000 is not permitted.

VH, bits [11:8]

Virtualization Host Extensions. Defined values are:

| VH | Meaning |
|--------|---|
| 0b0000 | Virtualization Host Extensions not supported. |
| 0b0001 | Virtualization Host Extensions supported. |

All other values are reserved.

FEAT_VHE implements the functionality identified by the value 0b0001.

From Armv8.1, the only permitted value is 0b0001.

VMIDBits, bits [7:4]

Number of VMID bits. Defined values are:

| VMIDBits | Meaning |
|----------|---------|
| 0b0000 | 8 bits |
| 0b0010 | 16 bits |

All other values are reserved.

FEAT_VMID16 implements the functionality identified by the value 0b0010.

From Armv8.1, the permitted values are 0b0000 and 0b0010.

HAFDBS, bits [3:0]

Hardware updates to Access flag and Dirty state in translation tables. Defined values are:

| HAFDBS | Meaning |
|--------|---|
| 0b0000 | Hardware update of the Access flag and dirty state are not supported. |
| 0b0001 | Hardware update of the Access flag is supported. |
| 0b0010 | Hardware update of both the Access flag and dirty state is supported. |

All other values are reserved.

FEAT_HAFDBS implements the functionality identified by the values 0b0001 and 0b0010.

From Armv8.1, the permitted values are 0b0000, 0b0001, and 0b0010.

Accessing ID_AA64MMFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64MMFR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0111 | 0b001 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64MMFR1_EL1;
elseif PSTATE.EL == EL2 then
    return ID_AA64MMFR1_EL1;
elseif PSTATE.EL == EL3 then
    return ID_AA64MMFR1_EL1;

```


ID_AA64MMFR2_EL1, AArch64 Memory Model Feature Register 2

The ID_AA64MMFR2_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64MMFR2_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-----|----|----|----|-------|----|----|----|---------|----|----|----|------|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| EOPD | | | | EVT | | | | BBM | | | | TTL | | | | RES0 | | | | FWB | | | | IDS | | | | AT | | | |
| ST | | | | NV | | | | CCIDX | | | | VARange | | | | IESB | | | | LSM | | | | UAO | | | | CnP | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

EOPD, bits [63:60]

Indicates support for the EOPD mechanism. Defined values are:

| EOPD | Meaning |
|--------|-------------------------------------|
| 0b0000 | EOPDx mechanism is not implemented. |
| 0b0001 | EOPDx mechanism is implemented. |

All other values are reserved.

FEAT_EOPD implements the functionality identified by the value 0b0001.

In Armv8.4, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

If FEAT_EOPD is implemented, FEAT_CSV3 must be implemented.

EVT, bits [59:56]

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the [HCR_EL2](#).{TTLBOS, TLBIS, TOCU, TICAB, TID4} traps. Defined values are:

| EVT | Meaning |
|------------|---|
| 0b0000 | HCR_EL2 .{TTLBOS, TTLBIS, TOCU, TICAB, TID4} traps are not supported. |
| 0b0001 | HCR_EL2 .{TOCU, TICAB, TID4} traps are supported. HCR_EL2 .{TTLBOS, TTLBIS} traps are not supported. |
| 0b0010 | HCR_EL2 .{TTLBOS, TTLBIS, TOCU, TICAB, TID4} traps are supported. |

All other values are reserved.

FEAT_EVT implements the functionality identified by the values 0b0001 and 0b0010.

If EL2 is not implemented, the only permitted value is 0b0000.

In Armv8.2, the permitted values are 0b0000, 0b0001, and 0b0010.

From Armv8.5, the permitted values are:

- 0b0000 when EL2 is not implemented.
- 0b0010 when EL2 is implemented.

BBM, bits [55:52]

Allows identification of the requirements of the hardware to have break-before-make sequences when changing block size for a translation.

| BBM | Meaning |
|------------|---|
| 0b0000 | Level 0 support for changing block size is supported. |
| 0b0001 | Level 1 support for changing block size is supported. |
| 0b0010 | Level 2 support for changing block size is supported. |

All other values are reserved.

FEAT_BBIM implements the functionality identified by the values 0b0000, 0b0001, and 0b0010.

From Armv8.4, the permitted values are 0b0000, 0b0001, and 0b0010.

TTL, bits [51:48]

Indicates support for TTL field in address operations. Defined values are:

| TTL | Meaning |
|------------|---|
| 0b0000 | TLB maintenance instructions by address have bits[47:44] as RES0. |
| 0b0001 | TLB maintenance instructions by address have bits[47:44] holding the TTL field. |

All other values are reserved.

FEAT_TTL implements the functionality identified by the value 0b0001.

This field affects [TLBI IPAS2E1](#), [TLBI IPAS2E1IS](#), [TLBI IPAS2E1OS](#), [TLBI IPAS2LE1](#), [TLBI IPAS2LE1IS](#), [TLBI IPAS2LE1OS](#), [TLBI VAAE1](#), [TLBI VAAE1IS](#), [TLBI VAAE1OS](#), [TLBI VAALE1](#), [TLBI VAALE1IS](#), [TLBI VAALE1OS](#), [TLBI VAE1](#), [TLBI VAE1IS](#), [TLBI VAE1OS](#), [TLBI VAE2](#), [TLBI VAE2IS](#), [TLBI VAE2OS](#), [TLBI VAE3](#), [TLBI VAE3IS](#), [TLBI VAE3OS](#), [TLBI VALE1](#), [TLBI VALE1IS](#), [TLBI VALE1OS](#), [TLBI VALE2](#), [TLBI VALE2IS](#), [TLBI VALE2OS](#), [TLBI VALE3](#), [TLBI VALE3IS](#), [TLBI VALE3OS](#).

From Armv8.4, the only permitted value is 0b0001.

Bits [47:44]

Reserved, RES0.

FWB, bits [43:40]

Indicates support for [HCR_EL2](#).FWB. Defined values are:

| FWB | Meaning |
|------------|--|
| 0b0000 | HCR_EL2 .FWB bit is not supported. |
| 0b0001 | HCR_EL2 .FWB is supported. |

All other values reserved.

FEAT_S2FWB implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0001.

IDS, bits [39:36]

Indicates the value of ESR_ELx.EC that reports an exception generated by a read access to the feature ID space. Defined values are:

| IDS | Meaning |
|------------|---|
| 0b0000 | An exception which is generated by a read access to the feature ID space, other than a trap caused by HCR_EL2 .TIDx, SCTLR_EL1 .UCT, or SCTLR_EL2 .UCT, is reported by ESR_ELx.EC == 0x0. |
| 0b0001 | All exceptions generated by an AArch64 read access to the feature ID space are reported by ESR_ELx.EC == 0x18. |

All other values are reserved.

The Feature ID space is defined as the System register space in AArch64 with op0==3, op1=={0, 1, 3}, CRn==0, CRm=={0-7}, op2=={0-7}.

FEAT_IDST implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0001.

AT, bits [35:32]

Identifies support for unaligned single-copy atomicity and atomic functions. Defined values are:

| AT | Meaning |
|-----------|--|
| 0b0000 | Unaligned single-copy atomicity and atomic functions are not supported. |
| 0b0001 | Unaligned single-copy atomicity and atomic functions with a 16-byte address range aligned to 16-bytes are supported. |

All other values are reserved.

FEAT_LSE2 implements the functionality identified by the value 0b0001.

In Armv8.2, the permitted values are 0b0000 and 0b0001.

From Armv8.4, the only permitted value is 0b0001.

ST, bits [31:28]

Identifies support for small translation tables. Defined values are:

| ST | Meaning |
|-----------|--|
| 0b0000 | The maximum value of the TCR_ELx.{T0SZ,T1SZ} and VTCR_EL2.T0SZ fields is 39. |
| 0b0001 | The maximum value of the TCR_ELx.{T0SZ,T1SZ} and VTCR_EL2.T0SZ fields is 48 for 4KB and 16KB granules, and 47 for 64KB granules. |

All other values are reserved.

FEAT_TTST implements the functionality identified by the value 0b0001.

If FEAT_SEL2 is implemented, the only permitted value is 0b0001.

In an implementation which does not support FEAT_SEL2, the permitted values are 0b0000 and 0b0001.

NV, bits [27:24]

Nested Virtualization. If EL2 is implemented, indicates support for the use of nested virtualization. Defined values are:

| NV | Meaning |
|-----------|---|
| 0b0000 | Nested virtualization is not supported. |
| 0b0001 | The HCR_EL2 .{AT, NV1, NV} bits are implemented. |
| 0b0010 | The VNCR_EL2 register and the HCR_EL2 .{NV2, AT, NV1, NV} bits are implemented. |

All other values are reserved.

If EL2 is not implemented, the only permitted value is 0b0000.

FEAT_NV implements the functionality identified by the value 0b0001.

FEAT_NV2 implements the functionality identified by the value 0b0010.

In Armv8.3, if EL2 is implemented, the permitted values are 0b0000 and 0b0001.

From Armv8.4, if EL2 is implemented, the permitted values are 0b0000, 0b0001, and 0b0010.

CCIDX, bits [23:20]

Support for the use of revised [CCSIDR_EL1](#) register format. Defined values are:

| CCIDX | Meaning |
|--------------|--|
| 0b0000 | 32-bit format implemented for all levels of the CCSIDR_EL1 . |
| 0b0001 | 64-bit format implemented for all levels of the CCSIDR_EL1 . |

All other values are reserved.

FEAT_CCIDX implements the functionality identified by the value 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

VARange, bits [19:16]

Indicates support for a larger virtual address. Defined values are:

| VARange | Meaning |
|----------------|--|
| 0b0000 | VMSAv8-64 supports 48-bit VAs. |
| 0b0001 | VMSAv8-64 supports 52-bit VAs when using the 64KB translation granule. The size for other translation granules is not defined by this field. |

All other values are reserved.

FEAT_LVA implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

IESB, bits [15:12]

Indicates support for the IESB bit in the SCTLR_ELx registers. Defined values are:

| IESB | Meaning |
|-------------|---|
| 0b0000 | IESB bit in the SCTLR_ELx registers is not supported. |
| 0b0001 | IESB bit in the SCTLR_ELx registers is supported. |

All other values are reserved.

FEAT_IESB implements the functionality identified by the value 0b0001.

LSM, bits [11:8]

Indicates support for LSMAOE and nTLSMD bits in [SCTLR_EL1](#) and [SCTLR_EL2](#). Defined values are:

| LSM | Meaning |
|--------|---------------------------------------|
| 0b0000 | LSMAOE and nTLSMD bits not supported. |
| 0b0001 | LSMAOE and nTLSMD bits supported. |

All other values are reserved.

FEAT_LSMAOC implements the functionality identified by the value 0b0001.

UAO, bits [7:4]

User Access Override. Defined values are:

| UAO | Meaning |
|--------|--------------------|
| 0b0000 | UAO not supported. |
| 0b0001 | UAO supported. |

All other values are reserved.

FEAT_UAO implements the functionality identified by the value 0b0001.

From Armv8.2, the only permitted value is 0b0001.

CnP, bits [3:0]

Indicates support for Common not Private translations. Defined values are:

| CnP | Meaning |
|--------|--|
| 0b0000 | Common not Private translations not supported. |
| 0b0001 | Common not Private translations supported. |

All other values are reserved.

FEAT_TTCNP implements the functionality identified by the value 0b0001.

From Armv8.2, the only permitted value is 0b0001.

Accessing ID_AA64MMFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64MMFR2_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0111 | 0b010 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && (!IsZero(ID_AA64MMFR2_EL1) || boolean IMPLEMENTATION_DEFINED
"ID_AA64MMFR2_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64MMFR2_EL1;
elseif PSTATE.EL == EL2 then
    return ID_AA64MMFR2_EL1;
elseif PSTATE.EL == EL3 then
    return ID_AA64MMFR2_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64PFR0_EL1, AArch64 Processor Feature Register 0

The ID_AA64PFR0_EL1 characteristics are:

Purpose

Provides additional information about implemented PE features in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

The external register [EDPFR](#) gives information from this register.

Attributes

ID_AA64PFR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|----|----|----|----|----------------------|----|----|----|-------------------------|----|----|---------------------|----|----|----|---------------------|----|----|----|----------------------|----|----|----------------------|----|----|----|----|---------------------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| CSV3 | | | | | CSV2 | | | | RME | | | DIT | | | | AMU | | | | MPAM | | | SEL2 | | | | | SVE | | | |
| RAS | | | | | GIC | | | | AdvSIMD | | | FP | | | | EL3 | | | | EL2 | | | EL1 | | | | | EL0 | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

CSV3, bits [63:60]

Speculative use of faulting data. Defined values are:

| CSV3 | Meaning |
|--------|--|
| 0b0000 | This PE does not disclose whether data loaded under speculation with a permission or domain fault can be used to form an address or generate condition codes or SVE predicate values to be used by other instructions in the speculative sequence. |
| 0b0001 | Data loaded under speculation with a permission or domain fault cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence. The execution timing of any other instructions in the speculative sequence is not a function of the data loaded under speculation. |

All other values are reserved.

FEAT_CSV3 implements the functionality identified by the value 0b0001.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

If FEAT_E0PD is implemented, FEAT_CSV3 must be implemented.

CSV2, bits [59:56]

Speculative use of out of context branch targets. Defined values are:

| CSV2 | Meaning |
|--------|---|
| 0b0000 | This PE does not disclose whether branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context. |
| 0b0001 | Branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context only in a hard-to-determine way. Contexts do not include the SCXTNUM_ELx register contexts. Support for the SCXTNUM_ELx registers is defined in ID_AA64PFR1_EL1.CSV2_frac . |
| 0b0010 | Branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context only in a hard-to-determine way. The SCXTNUM_ELx registers are supported and the contexts include the SCXTNUM_ELx register contexts. |

All other values are reserved.

FEAT_CSV2 implements the functionality identified by the value 0b0001.

FEAT_CSV2_2 implements the functionality identified by the value 0b0010.

In Armv8.0, the permitted values are 0b0000, 0b0001, and 0b0010.

From Armv8.5, the permitted values are 0b0001 and 0b0010.

RME, bits [55:52]

Realm Management Extension (RME). Defined values are:

| RME | Meaning |
|--------|---|
| 0b0000 | Realm Management Extension not implemented. |
| 0b0001 | RMEv1 is implemented. |

All other values are reserved.

FEAT_RME implements the functionality identified by the value 0b0001.

DIT, bits [51:48]

Data Independent Timing. Defined values are:

| DIT | Meaning |
|--------|---|
| 0b0000 | AArch64 does not guarantee constant execution time of any instructions. |
| 0b0001 | AArch64 provides the PSTATE.DIT mechanism to guarantee constant execution time of certain instructions. |

All other values are reserved.

FEAT_DIT implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0001.

AMU, bits [47:44]

Indicates support for Activity Monitors Extension. Defined values are:

| AMU | Meaning |
|--------|--|
| 0b0000 | Activity Monitors Extension is not implemented. |
| 0b0001 | FEAT_AMUv1 is implemented. |
| 0b0010 | FEAT_AMUv1p1 is implemented. As 0b0001 and adds support for virtualization of the activity monitor event counters. |

All other values are reserved.

FEAT_AMUv1 implements the functionality identified by the value 0b0001.

FEAT_AMUv1p1 implements the functionality identified by the value 0b0010.

In Armv8.0, the only permitted value is 0b0000.

In Armv8.4, the permitted values are 0b0000 and 0b0001.

From Armv8.6, the permitted values are 0b0000, 0b0001, and 0b0010.

MPAM, bits [43:40]

Indicates support for MPAM Extension. Defined values are:

| MPAM | Meaning |
|--------|---|
| 0b0000 | If ID_AA64PFR1_EL1.MPAM_frac == 0b0000 , MPAM Extension is not implemented. |
| | If ID_AA64PFR1_EL1.MPAM_frac == 0b0001 , MPAM Extension version 0.1 is implemented. |
| 0b0001 | If ID_AA64PFR1_EL1.MPAM_frac == 0b0000 , MPAM Extension version 1.0 is implemented. |
| | If ID_AA64PFR1_EL1.MPAM_frac == 0b0001 , MPAM Extension version 1.1 is implemented. |

All other values are reserved.

SEL2, bits [39:36]

Secure EL2. Defined values are:

| SEL2 | Meaning |
|--------|--------------------------------|
| 0b0000 | Secure EL2 is not implemented. |
| 0b0001 | Secure EL2 is implemented. |

All other values are reserved.

FEAT_SEL2 implements the functionality identified by the value 0b0001.

SVE, bits [35:32]

Scalable Vector Extension. Defined values are:

| SVE | Meaning |
|--------|---|
| 0b0000 | SVE architectural state and programmers' model are not implemented. |
| 0b0001 | SVE architectural state and programmers' model are implemented. |

All other values are reserved.

If implemented, refer to [ID_AA64ZFR0_EL1](#) for information about which SVE instructions are available.

RAS, bits [31:28]

RAS Extension version. Defined values are:

| RAS | Meaning |
|------------|--|
| 0b0000 | No RAS Extension. |
| 0b0001 | RAS Extension implemented. |
| 0b0010 | FEAT_RASv1p1 implemented and, if EL3 is implemented, FEAT_DoubleFault implemented. As 0b0001, and adds support for: <ul style="list-style-type: none"> If EL3 is implemented, FEAT_DoubleFault. Additional ERXMISC<m>_EL1 System registers. Additional System registers ERXPFGCDN_EL1, ERXPFGCTL_EL1, and ERXPFGF_EL1, and the SCR_EL3.FIEN and HCR_EL2.FIEN trap controls, to support the optional RAS Common Fault Injection Model Extension. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS and support for the optional RAS Timestamp and RAS Common Fault Injection Model Extensions. |

All other values are reserved.

FEAT_RAS implements the functionality identified by the value 0b0001.

FEAT_RASv1p1 and FEAT_DoubleFault implement the functionality identified by the value 0b0010.

In Armv8.0 and Armv8.1, the permitted values are 0b0000 and 0b0001.

In Armv8.2, the only permitted value is 0b0001.

From Armv8.4, if FEAT_DoubleFault is implemented, the only permitted value is 0b0010.

From Armv8.4, when FEAT_DoubleFault is not implemented, and [ERRIDR_EL1](#) is 0, the permitted values are IMPLEMENTATION DEFINED 0b0001 or 0b0010.

Note

When the value of this field is 0b0001, [ID_AA64PFR1_EL1](#).RAS_frac indicates whether FEAT_RASv1p1 is implemented.

GIC, bits [27:24]

System register GIC CPU interface. Defined values are:

| GIC | Meaning |
|------------|--|
| 0b0000 | GIC CPU interface system registers not implemented. |
| 0b0001 | System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported. |
| 0b0011 | System register interface to version 4.1 of the GIC CPU interface is supported. |

All other values are reserved.

AdvSIMD, bits [23:20]

Advanced SIMD. Defined values are:

| AdvSIMD | Meaning |
|----------------|--|
| 0b0000 | Advanced SIMD is implemented, including support for the following Sisd and SIMD operations: <ul style="list-style-type: none"> Integer byte, halfword, word and doubleword element operations. Single-precision and double-precision floating-point arithmetic. Conversions between single-precision and half-precision data types, and double-precision and half-precision data types. |
| 0b0001 | As for 0b0000, and also includes support for half-precision floating-point arithmetic. |
| 0b1111 | Advanced SIMD is not implemented. |

All other values are reserved.

This field must have the same value as the FP field.

The permitted values are:

- 0b0000 in an implementation with Advanced SIMD support that does not include the FEAT_FP16 extension.
- 0b0001 in an implementation with Advanced SIMD support that includes the FEAT_FP16 extension.
- 0b1111 in an implementation without Advanced SIMD support.

FP, bits [19:16]

Floating-point. Defined values are:

| FP | Meaning |
|-----------|---|
| 0b0000 | Floating-point is implemented, and includes support for: <ul style="list-style-type: none"> Single-precision and double-precision floating-point types. Conversions between single-precision and half-precision data types, and double-precision and half-precision data types. |
| 0b0001 | As for 0b0000, and also includes support for half-precision floating-point arithmetic. |
| 0b1111 | Floating-point is not implemented. |

All other values are reserved.

This field must have the same value as the AdvSIMD field.

The permitted values are:

- 0b0000 in an implementation with floating-point support that does not include the FEAT_FP16 extension.
- 0b0001 in an implementation with floating-point support that includes the FEAT_FP16 extension.
- 0b1111 in an implementation without floating-point support.

EL3, bits [15:12]

EL3 Exception level handling. Defined values are:

| EL3 | Meaning |
|------------|---|
| 0b0000 | EL3 is not implemented. |
| 0b0001 | EL3 can be executed in AArch64 state only. |
| 0b0010 | EL3 can be executed in either AArch64 or AArch32 state. |

All other values are reserved.

EL2, bits [11:8]

EL2 Exception level handling. Defined values are:

| EL2 | Meaning |
|--------|---|
| 0b0000 | EL2 is not implemented. |
| 0b0001 | EL2 can be executed in AArch64 state only. |
| 0b0010 | EL2 can be executed in either AArch64 or AArch32 state. |

All other values are reserved.

EL1, bits [7:4]

EL1 Exception level handling. Defined values are:

| EL1 | Meaning |
|--------|---|
| 0b0001 | EL1 can be executed in AArch64 state only. |
| 0b0010 | EL1 can be executed in either AArch64 or AArch32 state. |

All other values are reserved.

EL0, bits [3:0]

EL0 Exception level handling. Defined values are:

| EL0 | Meaning |
|--------|---|
| 0b0001 | EL0 can be executed in AArch64 state only. |
| 0b0010 | EL0 can be executed in either AArch64 or AArch32 state. |

All other values are reserved.

Accessing ID_AA64PFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64PFR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_AA64PFR0_EL1;
    elsif PSTATE.EL == EL2 then
        return ID_AA64PFR0_EL1;
    elsif PSTATE.EL == EL3 then
        return ID_AA64PFR0_EL1;

```

ID_AA64PFR1_EL1, AArch64 Processor Feature Register 1

The ID_AA64PFR1_EL1 characteristics are:

Purpose

Reserved for future expansion of information about implemented PE features in AArch64 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

There are no configuration notes.

Attributes

ID_AA64PFR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|-----|----|----|----|------|----|----|----|-----------|----|----|----|----------|----|----|----|-----|----|----|----|------|----|----|----|-----------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | NMI | | | | CSV2_frac | | | |
| RNDR_trap | | | | SME | | | | RES0 | | | | MPAM_frac | | | | RAS_frac | | | | MTE | | | | SSBS | | | | BT | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:40]

Reserved, RES0.

NMI, bits [39:36]

Non-maskable Interrupt. Indicates support for Non-maskable interrupts. Defined values are:

| NMI | Meaning |
|--------|--|
| 0b0000 | SCTLR_ELx.{SPINTMASK, NMI} and PSTATE.ALLINT with its associated instructions are not supported. |
| 0b0001 | SCTLR_ELx.{SPINTMASK, NMI} and PSTATE.ALLINT with its associated instructions are supported. |

All other values are reserved.

FEAT_NMI implements the functionality identified by the value 0b0001.

From Armv8.8, the only permitted value is 0b0001.

CSV2_frac, bits [35:32]

CSV2 fractional field. Defined values are:

| CSV2_frac | Meaning |
|-----------|--|
| 0b0000 | This PE does not disclose whether branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context. The SCXTNUM_ELx registers are not supported. |
| 0b0001 | If ID_AA64PFR0_EL1.CSV2 is 0b0001, branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context only in a hard-to-determine way. Within a hardware-described context, branch targets trained for branches situated at one address can control speculative execution of branches situated at different addresses only in a hard-to-determine way. The SCXTNUM_ELx registers are not supported and the contexts do not include the SCXTNUM_ELx register contexts. |
| 0b0010 | If ID_AA64PFR0_EL1.CSV2 is 0b0001, branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context only in a hard-to-determine way. Within a hardware-described context, branch targets trained for branches situated at one address can control speculative execution of branches situated at different addresses only in a hard-to-determine way. The SCXTNUM_ELx registers are supported, but the contexts do not include the SCXTNUM_ELx register contexts. |

All other values are reserved.

FEAT_CSV2_1p1 implements the functionality identified by the value 0b0001.

FEAT_CSV2_1p2 implements the functionality identified by the value 0b0010.

From Armv8.0, the permitted values are 0b0000, 0b0001, and 0b0010.

This field is valid only if [ID_AA64PFR0_EL1.CSV2](#) is 0b0001.

RNDR_trap, bits [31:28]

Random Number trap to EL3 field. Defined values are:

| RNDR_trap | Meaning |
|-----------|--|
| 0b0000 | Trapping of RNDR and RNDRRS to EL3 is not supported. |
| 0b0001 | Trapping of RNDR and RNDRRS to EL3 is supported. SCR_EL3.TRNDR is present. |

All other values are reserved.

FEAT_RNG_TRAP implements the functionality identified by the value 0b0001.

SME, bits [27:24]

Scalable Matrix Extension field. Defined values are:

| SME | Meaning |
|--------|---|
| 0b0000 | SME architectural state and programmers' model are not implemented. |
| 0b0001 | SME architectural state and programmers' model are implemented. |

All other values are reserved.

FEAT_SME implements the functionality identified by the value 0b0001.

From Armv9.2, the permitted values are 0b0000 and 0b0001.

Bits [23:20]

Reserved, RES0.

MPAM_frac, bits [19:16]

MPAM Extension fractional field. Defined values are:

| MPAM_frac | Meaning |
|-----------|---|
| 0b0000 | If ID_AA64PFR0_EL1 .MPAM == 0b0000, MPAM Extension not implemented. |
| 0b0001 | If ID_AA64PFR0_EL1 .MPAM == 0b0001, MPAM Extension v1.0 is implemented. |
| 0b0001 | If ID_AA64PFR0_EL1 .MPAM == 0b0000, implements MPAM v0.1, which is like v1.1 but reduces support for Secure PARTIDs. |
| 0b0001 | If ID_AA64PFR0_EL1 .MPAM == 0b0001, implements MPAM v1.1 and adds support for MPAM2_EL2 .TIDR to provide trapping of MPAMIDR_EL1 when MPAMHCR_EL2 is not present. |

All other values are reserved.

RAS_frac, bits [15:12]

RAS Extension fractional field. Defined values are:

| RAS_frac | Meaning |
|----------|--|
| 0b0000 | If ID_AA64PFR0_EL1 .RAS == 0b0001, RAS Extension implemented. |
| 0b0001 | If ID_AA64PFR0_EL1 .RAS == 0b0001, as 0b0000 and adds support for: <ul style="list-style-type: none"> Additional ERXMISC<m>_EL1 System registers. Additional System registers ERXPFPCDN_EL1, ERXPFPCCTL_EL1, and ERXPFGF_EL1, and the SCR_EL3.FIEN and HCR_EL2.FIEN trap controls, to support the optional RAS Common Fault Injection Model Extension. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS , and support for the optional RAS Timestamp and RAS Common Fault Injection Model Extensions. |

All other values are reserved.

FEAT_RASv1p1 implements the functionality identified by the value 0b0001.

This field is valid only if [ID_AA64PFR0_EL1](#).RAS == 0b0001.

MTE, bits [11:8]

Support for the Memory Tagging Extension. Defined values are:

| MTE | Meaning |
|--------|---|
| 0b0000 | Memory Tagging Extension is not implemented. |
| 0b0001 | Instruction-only Memory Tagging Extension is implemented. |
| 0b0010 | Full Memory Tagging Extension is implemented. |
| 0b0011 | Memory Tagging Extension is implemented with support for asymmetric Tag Check Fault handling. |

All other values are reserved.

FEAT_MTE implements the functionality identified by the value 0b0001.

FEAT_MTE2 implements the functionality identified by the value 0b0010.

FEAT_MTE3 implements the functionality identified by the value 0b0011.

In Armv8.5, the permitted values are 0b0000, 0b0001 and 0b0010.

From Armv8.7, the value 0b0001 is not permitted.

SSBS, bits [7:4]

Speculative Store Bypassing controls in AArch64 state. Defined values are:

| SSBS | Meaning |
|--------|--|
| 0b0000 | AArch64 provides no mechanism to control the use of Speculative Store Bypassing. |
| 0b0001 | AArch64 provides the PSTATE.SSBS mechanism to mark regions that are Speculative Store Bypass Safe. |
| 0b0010 | As 0b0001, and adds the MSR and MRS instructions to directly read and write the PSTATE.SSBS field. |

All other values are reserved.

FEAT_SSBS implements the functionality identified by the value 0b0001.

FEAT_SSBS2 implements the functionality identified by the value 0b0010.

BT, bits [3:0]

Branch Target Identification mechanism support in AArch64 state. Defined values are:

| BT | Meaning |
|--------|--|
| 0b0000 | The Branch Target Identification mechanism is not implemented. |
| 0b0001 | The Branch Target Identification mechanism is implemented. |

All other values are reserved.

FEAT_BTI implements the functionality identified by the value 0b0001.

From Armv8.5, the only permitted value is 0b0001.

Accessing ID_AA64PFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64PFR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0100 | 0b001 |

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64PFR1_EL1;
elseif PSTATE.EL == EL2 then
    return ID_AA64PFR1_EL1;
elseif PSTATE.EL == EL3 then
    return ID_AA64PFR1_EL1;
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64SMFR0_EL1, SME Feature ID register 0

The ID_AA64SMFR0_EL1 characteristics are:

Purpose

Provides information about the implemented features of the AArch64 Scalable Matrix Extension, when the [ID_AA64PFR1_EL1](#).SME field is not zero.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64SMFR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|--------|--------|------|--------|------|----|----|----|----|----|-------|--------|--------|------|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| FA64 | RES0 | SMEver | I16I64 | RES0 | F64F64 | RES0 | | | | | | I8I32 | F16F32 | B16F32 | RES0 | F32F32 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

FA64, bit [63]

Indicates support for execution of the full A64 instruction set when the PE is in Streaming SVE mode. Defined values are:

| FA64 | Meaning |
|------|--|
| 0b0 | Only those A64 instructions defined as being legal can be executed in Streaming SVE mode. |
| 0b1 | All implemented A64 instructions can be executed in Streaming SVE mode, when enabled at the current Exception level by SMCR_EL1 .FA64, SMCR_EL2 .FA64, and SMCR_EL3 .FA64. |

FEAT_SME_FA64 implements the functionality identified by the value 0b1.

Bits [62:60]

Reserved, RES0.

SMEver, bits [59:56]

Indicates support for SME instructions when [ID_AA64PFR1_EL1](#).SME is not zero. Defined values are:

| SMEver | Meaning |
|--------|--|
| 0b0000 | The non-optional SME instructions are implemented. |

All other values are reserved.

FEAT_SME implements the functionality identified by the value 0b0000, when [ID_AA64PFR1_EL1](#).SME is not zero.

From Armv9.2, the only permitted value is 0b0000.

I16I64, bits [55:52]

Indicates SME support for instructions that accumulate into 64-bit integer elements in the ZA array. Defined values are:

| I16I64 | Meaning |
|--------|--|
| 0b0000 | Instructions that accumulate into 64-bit integer elements in the ZA array are not implemented. |
| 0b1111 | The variants of the ADDHA, ADDVA, SMOPA, SMOPS, SUMOPA, SUMOPS, UMOPA, UMOPS, USMOPA, and USMOPS instructions that accumulate into 64-bit integer element tiles are implemented. |

All other values are reserved.

The only permitted values are 0b0000 and 0b1111.

Bits [51:49]

Reserved, RES0.

F64F64, bit [48]

Indicates SME support for instructions that accumulate into double-precision floating-point elements in the ZA array. Defined values are:

| F64F64 | Meaning |
|--------|---|
| 0b0 | Instructions that accumulate into double-precision floating-point elements in the ZA array are not implemented. |
| 0b1 | The variants of the FMOPA and FMOPS instructions that accumulate into double-precision element tiles are implemented. |

Bits [47:40]

Reserved, RES0.

I8I32, bits [39:36]

Indicates SME support for accumulating 8-bit integer outer products into 32-bit integer tiles. Defined values are:

| I8I32 | Meaning |
|--------|---|
| 0b0000 | Instructions that accumulate 8-bit outer products into 32-bit tiles are not implemented. |
| 0b1111 | The SMOPA, SMOPS, SUMOPA, SUMOPS, UMOPA, UMOPS, USMOPA, and USMOPS instructions that accumulate 8-bit outer products into 32-bit tiles are implemented. |

All other values are reserved.

If FEAT_SME is implemented, the only permitted value is 0b1111.

F16F32, bit [35]

Indicates SME support for accumulating half-precision floating-point outer products into single-precision floating-point tiles. Defined values are:

| F16F32 | Meaning |
|---------------|---|
| 0b0 | Instructions that accumulate FP16 outer products into FP32 tiles are not implemented. |
| 0b1 | The FMOPA and FMOPS instructions that accumulate half-precision outer products into single-precision tiles are implemented. |

If FEAT_SME is implemented, the only permitted value is 0b1.

B16F32, bit [34]

Indicates SME support for accumulating BFloat16 outer products into single-precision floating-point tiles. Defined values are:

| B16F32 | Meaning |
|---------------|--|
| 0b0 | Instructions that accumulate BFloat16 outer products into single-precision tiles are not implemented. |
| 0b1 | The BFMOA and BFMOPS instructions that accumulate BFloat16 outer products into single-precision tiles are implemented. |

If FEAT_SME is implemented, the only permitted value is 0b1.

Bit [33]

Reserved, RES0.

F32F32, bit [32]

Indicates SME support for accumulating single-precision floating-point outer products into single-precision floating-point tiles. Defined values are:

| F32F32 | Meaning |
|---------------|---|
| 0b0 | Instructions that accumulate single-precision outer products into single-precision tiles are not implemented. |
| 0b1 | The FMOPA and FMOPS instructions that accumulate single-precision outer products into single-precision tiles are implemented. |

If FEAT_SME is implemented, the only permitted value is 0b1.

Bits [31:0]

Reserved, RES0.

Accessing ID_AA64SMFR0_EL1

This register is read-only and can be accessed from EL1 and higher.

This register is only accessible from the AArch64 state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64SMFR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b000 | 0b0000 | 0b0100 | 0b101 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_AA64SMFR0_EL1;
    elseif PSTATE.EL == EL2 then
        return ID_AA64SMFR0_EL1;
    elseif PSTATE.EL == EL3 then
        return ID_AA64SMFR0_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_AA64ZFR0_EL1, SVE Feature ID register 0

The ID_AA64ZFR0_EL1 characteristics are:

Purpose

Provides additional information about the implemented features of the AArch64 Scalable Vector Extension instruction set, when either or both of [ID_AA64PFR0_EL1](#).SVE and [ID_AA64PFR1_EL1](#).SME are not zero.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_AA64ZFR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|----|----|-------|----|----|----|---------|----|----|----|------|----|----|----|-----|----|----|----|------|----|----|----|--------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | F64MM | | | | F32MM | | | | RES0 | | | | I8MM | | | | SM4 | | | | RES0 | | | | SHA3 | | | |
| RES0 | | | | | | | | BF16 | | | | BitPerm | | | | RES0 | | | | | | | | AES | | | | SVEver | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:60]

Reserved, RES0.

F64MM, bits [59:56]

Indicates support for SVE FP64 double-precision floating-point matrix multiplication instructions. Defined values are:

| F64MM | Meaning |
|--------|--|
| 0b0000 | FP64 matrix multiplication and related instructions are not implemented. |
| 0b0001 | FP64 variant of the FMMLA instruction, and LD1RO* instructions are implemented. The 128-bit element variations of TRN1, TRN2, UZP1, UZP2, ZIP1, and ZIP2 are also implemented. |

All other values are reserved.

When the PE is in Streaming SVE mode, software should ignore and treat as zero the value in this field.

FEAT_F64MM implements the functionality identified by 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

F32MM, bits [55:52]

Indicates support for the SVE FP32 single-precision floating-point matrix multiplication instruction. Defined values are:

| F32MM | Meaning |
|--------------|--|
| 0b0000 | FP32 matrix multiplication instruction is not implemented. |
| 0b0001 | FP32 variant of the FMMLA instruction is implemented. |

All other values are reserved.

When the PE is in Streaming SVE mode, software should ignore and treat as zero the value in this field.

FEAT_F32MM implements the functionality identified by 0b0001.

From Arm v8.2, the permitted values are 0b0000 and 0b0001.

Bits [51:48]

Reserved, RES0.

I8MM, bits [47:44]

Indicates support for SVE Int8 matrix multiplication instructions. Defined values are:

| I8MM | Meaning |
|-------------|--|
| 0b0000 | Int8 matrix multiplication instructions are not implemented. |
| 0b0001 | SMMLA, SUDOT, UMMLA, USMMLA, and USDOT instructions are implemented. |

All other values are reserved.

When the PE is in Streaming SVE mode, software should ignore and treat as zero the value in this field.

FEAT_I8MM implements the functionality identified by 0b0001.

When Advanced SIMD and SVE are both implemented, this field must return the same value as [ID_AA64ISAR1_EL1.I8MM](#).

From Armv8.6, if FEAT_SVE is implemented, then the only permitted value is 0b0001. Otherwise, the value can be 0b0000 or 0b0001.

SM4, bits [43:40]

Indicates support for SVE SM4 instructions. Defined values are:

| SM4 | Meaning |
|------------|--|
| 0b0000 | SVE SM4 instructions are not implemented. |
| 0b0001 | SVE SM4E and SM4EKEY instructions are implemented. |

All other values are reserved.

When the PE is in Streaming SVE mode, software should ignore and treat as zero the value in this field.

FEAT_SVE_SM4 implements the functionality identified by 0b0001.

Bits [39:36]

Reserved, RES0.

SHA3, bits [35:32]

Indicates support for the SVE SHA3 instructions. Defined values are:

| SHA3 | Meaning |
|--------|--|
| 0b0000 | SVE SHA3 instructions are not implemented. |
| 0b0001 | SVE RAX1 instruction is implemented. |

All other values are reserved.

When the PE is in Streaming SVE mode, software should ignore and treat as zero the value in this field.

FEAT_SVE_SHA3 implements the functionality identified by 0b0001.

Bits [31:24]

Reserved, RES0.

BF16, bits [23:20]

Indicates support for SVE BFloat16 instructions. Defined values are:

| BF16 | Meaning |
|--------|--|
| 0b0000 | BFloat16 instructions are not implemented. |
| 0b0001 | BFCVT, BFCVTNT, BFDOT, BFMLALB, BFMLALT, and BFMMMLA instructions are implemented. |
| 0b0010 | As 0b0001, but the FPCR .EBF field is also supported. |

All other values are reserved.

FEAT_BF16 implements the functionality identified by 0b0001.

FEAT_EBF16 implements the functionality identified by 0b0010.

When Advanced SIMD and SVE are both implemented, this field must return the same value as [ID_AA64ISAR1_EL1](#).BF16.

From Armv8.6, if FEAT_SME is implemented, the permitted values are 0b0001 and 0b0010.

From Armv8.6, if FEAT_SME is not implemented, the only permitted value is 0b0001.

BitPerm, bits [19:16]

Indicates support for SVE bit permute instructions. Defined values are:

| BitPerm | Meaning |
|---------|--|
| 0b0000 | SVE bit permute instructions are not implemented. |
| 0b0001 | SVE BDEP, BEXT, and BGRP instructions are implemented. |

All other values are reserved.

When the PE is in Streaming SVE mode, software should ignore and treat as zero the value in this field.

FEAT_SVE_BitPerm implements the functionality identified by 0b0001.

Bits [15:8]

Reserved, RES0.

AES, bits [7:4]

Indicates support for SVE AES instructions. Defined values are:

| AES | Meaning |
|--------|--|
| 0b0000 | SVE AES instructions are not implemented. |
| 0b0001 | SVE AESE, AESD, AESMC, and AESIMC instructions are implemented. |
| 0b0010 | As 0b0001, plus SVE PMULLB and PMULLT instructions with 64-bit source. |

All other values are reserved.

When the PE is in Streaming SVE mode, software should ignore and treat as zero the value in this field.

FEAT_SVE_AES implements the functionality identified by the value 0b0001.

FEAT_SVE_PMULL128 implements the functionality identified by the value 0b0010.

The permitted values are 0b0000 and 0b0010.

SVEver, bits [3:0]

Indicates support for SVE. Defined values are:

| SVEver | Meaning |
|--------|---|
| 0b0000 | SVE instructions are implemented. |
| 0b0001 | The SVE and non-optional SVE2 instructions are implemented. |

All other values are reserved.

FEAT_SVE2 and FEAT_SME implement the functionality identified by the value 0b0001.

From Armv9, if FEAT_SME is implemented, the only permitted value is 0b0001. This value indicates that SVE and SVE2 instructions are implemented when the PE is in Streaming SVE mode.

Note

Irrespective of the value of ID_AA64ZFR0_EL1.SVEver, when the PE is in Streaming SVE mode, software should not attempt to execute any of the SVE and SVE2 instructions that are illegal in Streaming SVE mode.

Accessing ID_AA64ZFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AA64ZFR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0100 | 0b100 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && (!IsZero(ID_AA64ZFR0_EL1) || boolean IMPLEMENTATION_DEFINED
"ID_AA64ZFR0_EL1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AA64ZFR0_EL1;
elseif PSTATE.EL == EL2 then
    return ID_AA64ZFR0_EL1;
elseif PSTATE.EL == EL3 then
    return ID_AA64ZFR0_EL1;

```


ID_AFR0_EL1, AArch32 Auxiliary Feature Register 0

The ID_AFR0_EL1 characteristics are:

Purpose

Provides information about the IMPLEMENTATION DEFINED features of the PE in AArch32 state.

Must be interpreted with the Main ID Register, [MIDR_EL1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_AFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_AFR0\[31:0\]](#).

Attributes

ID_AFR0_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------------|----|----|----|------------------------|----|----|----|------------------------|----|----|----|------------------------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | IMPLEMENTATION DEFINED | | | | IMPLEMENTATION DEFINED | | | | IMPLEMENTATION DEFINED | | | | IMPLEMENTATION DEFINED | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [15:12]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [11:8]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [7:4]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [3:0]

IMPLEMENTATION DEFINED.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_AFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_AFR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b011 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_AFR0_EL1;
elseif PSTATE.EL == EL2 then
    return ID_AFR0_EL1;
elseif PSTATE.EL == EL3 then
    return ID_AFR0_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_DFR0_EL1, AArch32 Debug Feature Register 0

The ID_DFR0_EL1 characteristics are:

Purpose

Provides top level information about the debug system in AArch32 state.

Must be interpreted with the Main ID Register, [MIDR_EL1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_DFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_DFR0\[31:0\]](#).

Attributes

ID_DFR0_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|---------|----|----|----|----------|----|----|----|---------|----|----|----|--------|----|----|----|---------|----|----|----|---------|----|----|----|--------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TraceFilt | | | | PerfMon | | | | MProfDbg | | | | MMapTrc | | | | CopTrc | | | | MMapDbg | | | | CopSDBG | | | | CopDbg | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

TraceFilt, bits [31:28]

Armv8.4 Self-hosted Trace Extension version. Defined values are:

| TraceFilt | Meaning |
|-----------|--|
| 0b0000 | Armv8.4 Self-hosted Trace Extension not implemented. |
| 0b0001 | Armv8.4 Self-hosted Trace Extension implemented. |

All other values are reserved.

FEAT_TRF implements the functionality added by the value 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

PerfMon, bits [27:24]

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version'

Defined values are:

| PerfMon | Meaning |
|---------|--|
| 0b0000 | Performance Monitors Extension not implemented. |
| 0b0001 | Performance Monitors Extension, PMUv1 implemented. |
| 0b0010 | Performance Monitors Extension, PMUv2 implemented. |
| 0b0011 | Performance Monitors Extension, PMUv3 implemented. |
| 0b0100 | PMUv3 for Armv8.1. As 0b0011, and adds support for: <ul style="list-style-type: none"> Extended 16-bit PMEVTYPER<n>.evtCount field. If EL2 is implemented, the HDCR.HPMD control. |
| 0b0101 | PMUv3 for Armv8.4. As 0b0100, and adds support for the PMMIR register. |
| 0b0110 | PMUv3 for Armv8.5. As 0b0101, and adds support for: <ul style="list-style-type: none"> 64-bit event counters. If EL2 is implemented, the HDCR.HCCD control. If EL3 is implemented, the MDCR_EL3.SCCD control. |
| 0b0111 | PMUv3 for Armv8.7. As 0b0110, and adds support for: <ul style="list-style-type: none"> The PMCR.FZO and, if EL2 is implemented, HDCR.HPMFZO controls. If EL3 is implemented, the MDCR_EL3.{MPMX,MCCD} controls. |
| 0b1000 | PMUv3 for Armv8.8. As 0b0111, and: <ul style="list-style-type: none"> Extends the Common event number space to include 0x0040 to 0x00BF and 0x4040 to 0x40BF. Removes the CONSTRAINED UNPREDICTABLE behaviors if a reserved or unimplemented PMU event number is selected. |
| 0b1111 | IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value for new implementations. |

All other values are reserved.

FEAT_PMUv3 implements the functionality identified by the value 0b0011.

FEAT_PMUv3p1 implements the functionality identified by the value 0b0100.

FEAT_PMUv3p4 implements the functionality identified by the value 0b0101.

FEAT_PMUv3p5 implements the functionality identified by the value 0b0110.

FEAT_PMUv3p7 implements the functionality identified by the value 0b0111.

FEAT_PMUv3p8 implements the functionality identified by the value 0b1000.

In any Armv8 implementation, the values 0b0001 and 0b0010 are not permitted.

From Armv8.1, if FEAT_PMUv3 is implemented, the value 0b0011 is not permitted.

From Armv8.4, if FEAT_PMUv3 is implemented, the value 0b0100 is not permitted.

From Armv8.5, if FEAT_PMUv3 is implemented, the value 0b0101 is not permitted.

From Armv8.7, if FEAT_PMUv3 is implemented, the value 0b0110 is not permitted.

From Armv8.8, if FEAT_PMUv3 is implemented, the value 0b0111 is not permitted.

Note

In Armv7, the value 0b0000 can mean that PMUv1 is implemented. PMUv1 is not permitted in an Armv8 implementation.

MProfDbg, bits [23:20]

M-profile Debug. Support for memory-mapped debug model for M-profile processors. Defined values are:

| MProfDbg | Meaning |
|----------|--|
| 0b0000 | Not supported. |
| 0b0001 | Support for M-profile Debug architecture, with memory-mapped access. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

MMapTrc, bits [19:16]

Memory-mapped Trace. Support for memory-mapped trace model. Defined values are:

| MMapTrc | Meaning |
|---------|--|
| 0b0000 | Not supported. |
| 0b0001 | Support for Arm trace architecture, with memory-mapped access. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

For more information, see the ARM® Embedded Trace Macrocell Architecture Specification, ETMv4 (ARM IHI 0064).

CopTrc, bits [15:12]

Support for System registers-based trace model, using registers in the coproc == 0b1110 encoding space. Defined values are:

| CopTrc | Meaning |
|--------|---|
| 0b0000 | Not supported. |
| 0b0001 | Support for Arm trace architecture, with System registers access. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

For more information, see the ARM® Embedded Trace Macrocell Architecture Specification, ETMv4 (ARM IHI 0064).

MMapDbg, bits [11:8]

Memory-mapped Debug. Support for Armv7 memory-mapped debug model for A and R-profile processors. Defined values are:

| MMapDbg | Meaning |
|---------|--|
| 0b0000 | Not supported. |
| 0b0100 | Support for Armv7, v7 Debug architecture, with memory-mapped access. |
| 0b0101 | Support for Armv7, v7.1 Debug architecture, with memory-mapped access. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

The optional memory map defined by Armv8 is not compatible with Armv7.

CopSDBG, bits [7:4]

Support for a System registers-based Secure debug model, using registers in the coproc = 0b1110 encoding space, for an A-profile processor that includes EL3.

If EL3 is not implemented and the implemented Security state is Non-secure state, this field is RES0. Otherwise, this field reads the same as bits [3:0].

CopDbg, bits [3:0]

Debug architecture version. Indicates presence of Armv8 debug architecture. Defined values are:

| CopDbg | Meaning |
|--------|---|
| 0b0000 | Not supported. |
| 0b0010 | Armv6, v6 Debug architecture, with System registers access. |
| 0b0011 | Armv6, v6.1 Debug architecture, with System registers access. |
| 0b0100 | Armv7, v7 Debug architecture, with System registers access. |
| 0b0101 | Armv7, v7.1 Debug architecture, with System registers access. |
| 0b0110 | Armv8 debug architecture. |
| 0b0111 | Armv8 debug architecture with Virtualization Host Extensions. |
| 0b1000 | Armv8.2 debug architecture, FEAT_Debugv8p2. |
| 0b1001 | Armv8.4 debug architecture, FEAT_Debugv8p4. |
| 0b1010 | Armv8.8 debug architecture, FEAT_Debugv8p8. |

All other values are reserved.

The values 0b0000, 0b0010, 0b0011, 0b0100, and 0b0101 are not permitted in Armv8.

FEAT_VHE adds the functionality identified by the value 0b0111.

FEAT_Debugv8p2 adds the functionality identified by the value 0b1000.

FEAT_Debugv8p4 adds the functionality identified by the value 0b1001.

FEAT_Debugv8p8 adds the functionality identified by the value 0b1010.

From Armv8.1, when FEAT_VHE is implemented the value 0b0110 is not permitted.

From Armv8.2, the values 0b0110 and 0b0111 are not permitted.

From Armv8.4, the value 0b1000 is not permitted.

From Armv8.8, the value 0b1001 is not permitted.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_DFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_DFR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b010 |

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_DFR0_EL1;
elseif PSTATE.EL == EL2 then
    return ID_DFR0_EL1;
elseif PSTATE.EL == EL3 then
    return ID_DFR0_EL1;
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_DFR1_EL1, Debug Feature Register 1

The ID_DFR1_EL1 characteristics are:

Purpose

Provides top level information about the debug system in AArch32.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_DFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_DFR1\[31:0\]](#).

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_DFR1_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|-------|----|----|----|-------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | RES0 | | | | | | | | HPMNO | | | | MTPMU | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:8]

Reserved, RES0.

HPMNO, bits [7:4]

Zero PMU event counters for a Guest operating system. Defined values are:

| HPMNO | Meaning |
|--------|--|
| 0b0000 | Setting HDCR .HPMN to zero has CONSTRAINED UNPREDICTABLE behavior. |
| 0b0001 | Setting HDCR .HPMN to zero has defined behavior. |

All other values are reserved.

If FEAT_PMUv3 is not implemented, FEAT_FGT is not implemented, or EL2 is not implemented, the only permitted value is 0b0000.

FEAT_HPMNO implements the functionality identified by the value 0b0001.

From Armv8.8, in an implementation that includes FEAT_PMUv3, FEAT_FGT, and EL2, the value 0b0000 is not permitted.

MTPMU, bits [3:0]

Multi-threaded PMU extension. Defined values are:

| MTPMU | Meaning |
|--------|--|
| 0b0000 | FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, it is IMPLEMENTATION DEFINED whether PMEVTYPER<n>. are read/write or RES0. |
| 0b0001 | FEAT_MTPMU and FEAT_PMUv3 implemented. PMEVTYPER<n>. are read/write. When FEAT_MTPMU is disabled, the Effective values of PMEVTYPER<n>. are 0. |
| 0b1111 | FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, PMEVTYPER<n>. are RES0. |

All other values are reserved.

FEAT_MTPMU implements the functionality identified by the value 0b0001.

From Armv8.6, in an implementation that includes FEAT_PMUv3, the value 0b0000 is not permitted.

In an implementation that does not include FEAT_PMUv3, the value 0b0001 is not permitted.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_DFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_DFR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0011 | 0b101 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && (!IsZero(ID_DFR1_EL1) || boolean IMPLEMENTATION_DEFINED "ID_DFR1_EL1
trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_DFR1_EL1;
    elseif PSTATE.EL == EL2 then
        return ID_DFR1_EL1;
    elseif PSTATE.EL == EL3 then
        return ID_DFR1_EL1;

```


ID_ISAR0_EL1, AArch32 Instruction Set Attribute Register 0

The ID_ISAR0_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR1_EL1](#), [ID_ISAR2_EL1](#), [ID_ISAR3_EL1](#), [ID_ISAR4_EL1](#), and [ID_ISAR5_EL1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_ISAR0\[31:0\]](#).

Attributes

ID_ISAR0_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|--------|----|----|----|-------|----|----|----|---------|----|----|----|-----------|----|----|----|----------|----|----|----|----------|----|----|----|------|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | Divide | | | | Debug | | | | Coprocc | | | | CmpBranch | | | | BitField | | | | BitCount | | | | Swap | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:28]

Reserved, RES0.

Divide, bits [27:24]

Indicates the implemented Divide instructions. Defined values are:

| Divide | Meaning |
|--------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds SDIV and UDIV in the T32 instruction set. |
| 0b0010 | As for 0b0001, and adds SDIV and UDIV in the A32 instruction set. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Debug, bits [23:20]

Indicates the implemented Debug instructions. Defined values are:

| Debug | Meaning |
|--------------|-------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds BKPT. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Coproc, bits [19:16]

Indicates the implemented System register access instructions. Defined values are:

| Coproc | Meaning |
|---------------|--|
| 0b0000 | None implemented, except for instructions separately attributed by the architecture to provide access to AArch32 System registers and System instructions. |
| 0b0001 | Adds generic CDP, LDC, MCR, MRC, and STC. |
| 0b0010 | As for 0b0001, and adds generic CDP2, LDC2, MCR2, MRC2, and STC2. |
| 0b0011 | As for 0b0010, and adds generic MCRR and MRRC. |
| 0b0100 | As for 0b0011, and adds generic MCRR2 and MRRC2. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

CmpBranch, bits [15:12]

Indicates the implemented combined Compare and Branch instructions in the T32 instruction set. Defined values are:

| CmpBranch | Meaning |
|------------------|--------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds CBNZ and CBZ. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

BitField, bits [11:8]

Indicates the implemented BitField instructions. Defined values are:

| BitField | Meaning |
|-----------------|--------------------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds BFC, BFI, SBFX, and UBFX. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

BitCount, bits [7:4]

Indicates the implemented Bit Counting instructions. Defined values are:

| BitCount | Meaning |
|-----------------|-------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds CLZ. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Swap, bits [3:0]

Indicates the implemented Swap instructions in the A32 instruction set. Defined values are:

| Swap | Meaning |
|--------|--------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds SWP and SWPB. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_ISAR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_ISAR0_EL1;
    elseif PSTATE.EL == EL2 then
        return ID_ISAR0_EL1;
    elseif PSTATE.EL == EL3 then
        return ID_ISAR0_EL1;

```

ID_ISAR1_EL1, AArch32 Instruction Set Attribute Register 1

The ID_ISAR1_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0_EL1](#), [ID_ISAR2_EL1](#), [ID_ISAR3_EL1](#), [ID_ISAR4_EL1](#), and [ID_ISAR5_EL1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_ISAR1\[31:0\]](#).

Attributes

ID_ISAR1_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|-----------|----|----|----|-----------|----|----|----|--------|----|----|----|--------|----|----|----|-----------|----|----|----|--------|----|----|----|--------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Jazelle | | | | Interwork | | | | Immediate | | | | IfThen | | | | Extend | | | | Except_AR | | | | Except | | | | Endian | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

Jazelle, bits [31:28]

Indicates the implemented Jazelle extension instructions. Defined values are:

| Jazelle | Meaning |
|---------|---|
| 0b0000 | No support for Jazelle. |
| 0b0001 | Adds the BXJ instruction and the J bit in the PSR. This setting might indicate a trivial implementation of the Jazelle extension. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Interwork, bits [27:24]

Indicates the implemented Interworking instructions. Defined values are:

| Interwork | Meaning |
|-----------|--|
| 0b0000 | None implemented. |
| 0b0001 | Adds the BX instruction, and the T bit in the PSR. |
| 0b0010 | As for 0b0001, and adds the BLX instruction. PC loads have BX-like behavior. |
| 0b0011 | As for 0b0010, and guarantees that data-processing instructions in the A32 instruction set with the PC as the destination and the S bit clear have BX-like behavior. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

Immediate, bits [23:20]

Indicates the implemented data-processing instructions with long immediates. Defined values are:

| Immediate | Meaning |
|-----------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds: <ul style="list-style-type: none"> The MOVT instruction. The MOV instruction encodings with zero-extended 16-bit immediates. The T32 ADD and SUB instruction encodings with zero-extended 12-bit immediates, and the other ADD, ADR, and SUB encodings cross-referenced by the pseudocode for those encodings. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

IfThen, bits [19:16]

Indicates the implemented If-Then instructions in the T32 instruction set. Defined values are:

| IfThen | Meaning |
|--------|--|
| 0b0000 | None implemented. |
| 0b0001 | Adds the IT instructions, and the IT bits in the PSRs. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Extend, bits [15:12]

Indicates the implemented Extend instructions. Defined values are:

| Extend | Meaning |
|--------|--|
| 0b0000 | No scalar sign-extend or zero-extend instructions are implemented, where scalar instructions means non-Advanced SIMD instructions. |
| 0b0001 | Adds the SXTB, SXTH, UXTB, and UXTH instructions. |
| 0b0010 | As for 0b0001, and adds the SXTB16, SXTAB, SXTAB16, SXTAH, UXTB16, UXTAB, UXTAB16, and UXTAH instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Except_AR, bits [11:8]

Indicates the implemented A and R-profile exception-handling instructions. Defined values are:

| Except_AR | Meaning |
|-----------|--|
| 0b0000 | None implemented. |
| 0b0001 | Adds the SRS and RFE instructions, and the A and R-profile forms of the CPS instruction. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Except, bits [7:4]

Indicates the implemented exception-handling instructions in the A32 instruction set. Defined values are:

| Except | Meaning |
|--------|--|
| 0b0000 | Not implemented. This indicates that the User bank and Exception return forms of the LDM and STM instructions are not implemented. |
| 0b0001 | Adds the LDM (exception return), LDM (user registers), and STM (user registers) instruction versions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Endian, bits [3:0]

Indicates the implemented Endian instructions. Defined values are:

| Endian | Meaning |
|--------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds the SETEND instruction, and the E bit in the PSRs. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_ISAR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_ISAR1_EL1;
    elsif PSTATE.EL == EL2 then
        return ID_ISAR1_EL1;
    elsif PSTATE.EL == EL3 then
        return ID_ISAR1_EL1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR2_EL1, AArch32 Instruction Set Attribute Register 2

The ID_ISAR2_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0_EL1](#), [ID_ISAR1_EL1](#), [ID_ISAR3_EL1](#), [ID_ISAR4_EL1](#), and [ID_ISAR5_EL1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_ISAR2\[31:0\]](#).

Attributes

ID_ISAR2_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|--------|----|----|----|-------|----|----|----|-------|----|----|----|------|----|----|----|----------------|----|----|----|---------|----|----|----|-----------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Reversal | | | | PSR_AR | | | | MultU | | | | MultS | | | | Mult | | | | MultiAccessInt | | | | MemHint | | | | LoadStore | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

Reversal, bits [31:28]

Indicates the implemented Reversal instructions. Defined values are:

| Reversal | Meaning |
|----------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds the REV, REV16, and REVSH instructions. |
| 0b0010 | As for 0b0001, and adds the RBIT instruction. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

PSR_AR, bits [27:24]

Indicates the implemented A and R-profile instructions to manipulate the PSR. Defined values are:

| PSR_AR | Meaning |
|---------------|--|
| 0b0000 | None implemented. |
| 0b0001 | Adds the MRS and MSR instructions, and the exception return forms of data-processing instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

The exception return forms of the data-processing instructions are:

- In the A32 instruction set, data-processing instructions with the PC as the destination and the S bit set. These instructions might be affected by the WithShifts attribute.
- In the T32 instruction set, the SUBS PC,LR,#N instruction.

MultU, bits [23:20]

Indicates the implemented advanced unsigned Multiply instructions. Defined values are:

| MultU | Meaning |
|--------------|--|
| 0b0000 | None implemented. |
| 0b0001 | Adds the UMULL and UMLAL instructions. |
| 0b0010 | As for 0b0001, and adds the UMAAL instruction. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

MultS, bits [19:16]

Indicates the implemented advanced signed Multiply instructions. Defined values are:

| MultS | Meaning |
|--------------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds the SMULL and SMLAL instructions. |
| 0b0010 | As for 0b0001, and adds the SMLABB, SMLABT, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMULBB, SMULBT, SMULTB, SMULTT, SMULWB, and SMULWT instructions. Also adds the Q bit in the PSRs. |
| 0b0011 | As for 0b0010, and adds the SMLAD, SMLADX, SMLALD, SMLALDX, SMLSD, SMLSDX, SMLSLD, SMLSLDX, SMMLA, SMMLAR, SMMLS, SMMLSR, SMMUL, SMMULR, SMUAD, SMUADX, SMUSD, and SMUSDX instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

Mult, bits [15:12]

Indicates the implemented additional Multiply instructions. Defined values are:

| Mult | Meaning |
|-------------|---|
| 0b0000 | No additional instructions implemented. This means only MUL is implemented. |
| 0b0001 | Adds the MLA instruction. |
| 0b0010 | As for 0b0001, and adds the MLS instruction. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

MultiAccessInt, bits [11:8]

Indicates the support for interruptible multi-access instructions. Defined values are:

| MultiAccessInt | Meaning |
|----------------|--|
| 0b0000 | No support. This means the LDM and STM instructions are not interruptible. |
| 0b0001 | LDM and STM instructions are restartable. |
| 0b0010 | LDM and STM instructions are continuable. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

MemHint, bits [7:4]

Indicates the implemented Memory Hint instructions. Defined values are:

| MemHint | Meaning |
|---------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds the PLD instruction. |
| 0b0010 | Adds the PLD instruction. (0b0001 and 0b0010 have identical effects.) |
| 0b0011 | As for 0b0001 (or 0b0010), and adds the PLI instruction. |
| 0b0100 | As for 0b0011, and adds the PLDW instruction. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0100.

LoadStore, bits [3:0]

Indicates the implemented additional load/store instructions. Defined values are:

| LoadStore | Meaning |
|-----------|--|
| 0b0000 | No additional load/store instructions implemented. |
| 0b0001 | Adds the LDRD and STRD instructions. |
| 0b0010 | As for 0b0001, and adds the Load Acquire (LDAB, LDAH, LDA, LDAEXB, LDAEXH, LDAEX, LDAEXD) and Store Release (STLB, STLH, STL, STLEXB, STLEXH, STLEX, STLEXD) instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_ISAR2_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_ISAR2_EL1;
elseif PSTATE.EL == EL2 then
    return ID_ISAR2_EL1;
elseif PSTATE.EL == EL3 then
    return ID_ISAR2_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR3_EL1, AArch32 Instruction Set Attribute Register 3

The ID_ISAR3_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0_EL1](#), [ID_ISAR1_EL1](#), [ID_ISAR2_EL1](#), [ID_ISAR4_EL1](#), and [ID_ISAR5_EL1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR3_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_ISAR3\[31:0\]](#).

Attributes

ID_ISAR3_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|---------|----|----|----|---------|----|----|----|-----------|----|----|----|-----------|----|----|----|-----|----|----|----|------|----|----|----|----------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| T32EE | | | | TrueNOP | | | | T32Copy | | | | TabBranch | | | | SynchPrim | | | | SVC | | | | SIMD | | | | Saturate | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

T32EE, bits [31:28]

Indicates the implemented T32EE instructions. Defined values are:

| T32EE | Meaning |
|--------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds the ENTERX and LEAVEX instructions, and modifies the load behavior to include null checking. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

TrueNOP, bits [27:24]

Indicates the implemented true NOP instructions. Defined values are:

| TrueNOP | Meaning |
|----------------|---|
| 0b0000 | None implemented. This means there are no NOP instructions that do not have any register dependencies. |
| 0b0001 | Adds true NOP instructions in both the T32 and A32 instruction sets. This also permits additional NOP-compatible hints. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

T32Copy, bits [23:20]

Indicates the support for T32 non flag-setting MOV instructions. Defined values are:

| T32Copy | Meaning |
|----------------|---|
| 0b0000 | Not supported. This means that in the T32 instruction set, encoding T1 of the MOV (register) instruction does not support a copy from a low register to a low register. |
| 0b0001 | Adds support for T32 instruction set encoding T1 of the MOV (register) instruction, copying from a low register to a low register. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

TabBranch, bits [19:16]

Indicates the implemented Table Branch instructions in the T32 instruction set. Defined values are:

| TabBranch | Meaning |
|------------------|------------------------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds the TBB and TBH instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

SynchPrim, bits [15:12]

Used in conjunction with ID_ISAR4.SynchPrim_frac to indicate the implemented Synchronization Primitive instructions. Defined values are:

| SynchPrim | Meaning |
|------------------|--|
| 0b0000 | If SynchPrim_frac == 0b0000, no Synchronization Primitives implemented. |
| 0b0001 | If SynchPrim_frac == 0b0000, adds the LDREX and STREX instructions. |
| | If SynchPrim_frac == 0b0011, also adds the CLREX, LDREXB, STREXB, and STREXH instructions. |
| 0b0010 | If SynchPrim_frac == 0b0000, as for [0b0001, 0b0011] and also adds the LDREXD and STREXD instructions. |

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A, the only permitted value is 0b0010.

SVC, bits [11:8]

Indicates the implemented SVC instructions. Defined values are:

| SVC | Meaning |
|------------|---------------------------|
| 0b0000 | Not implemented. |
| 0b0001 | Adds the SVC instruction. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

SIMD, bits [7:4]

Indicates the implemented SIMD instructions. Defined values are:

| SIMD | Meaning |
|--------|--|
| 0b0000 | None implemented. |
| 0b0001 | Adds the SSAT and USAT instructions, and the Q bit in the PSRs. |
| 0b0011 | As for 0b0001, and adds the PKHBT, PKHTB, QADD16, QADD8, QASX, QSUB16, QSUB8, QSAX, SADD16, SADD8, SASX, SEL, SHADD16, SHADD8, SHASX, SHSUB16, SHSUB8, SHSAX, SSAT16, SSUB16, SSUB8, SSAX, SXTAB16, SXTB16, UADD16, UADD8, UASX, UHADD16, UHADD8, UHASX, UHSUB16, UHSUB8, UHSAX, UQADD16, UQADD8, UQASX, UQSUB16, UQSUB8, UQSAX, USAD8, USADA8, USAT16, USUB16, USUB8, USAX, UXTAB16, and UXTB16 instructions. Also adds support for the GE[3:0] bits in the PSRs. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

The SIMD field relates only to implemented instructions that perform SIMD operations on the general-purpose registers. In an implementation that supports Advanced SIMD and floating-point instructions, [MVFR0](#), [MVFR1](#), and [MVFR2](#) give information about the implemented Advanced SIMD instructions.

Saturate, bits [3:0]

Indicates the implemented Saturate instructions. Defined values are:

| Saturate | Meaning |
|----------|--|
| 0b0000 | None implemented. This means no non-Advanced SIMD saturate instructions are implemented. |
| 0b0001 | Adds the QADD, QDADD, QDSUB, and QSUB instructions, and the Q bit in the PSRs. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR3_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_ISAR3_EL1

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b011 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_ISAR3_EL1;
    elsif PSTATE.EL == EL2 then
        return ID_ISAR3_EL1;
    elsif PSTATE.EL == EL3 then
        return ID_ISAR3_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR4_EL1, AArch32 Instruction Set Attribute Register 4

The ID_ISAR4_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0_EL1](#), [ID_ISAR1_EL1](#), [ID_ISAR2_EL1](#), [ID_ISAR3_EL1](#), and [ID_ISAR5_EL1](#).

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR4_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_ISAR4\[31:0\]](#).

Attributes

ID_ISAR4_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|-------|----|----|----|----------------|----|----|----|---------|----|----|----|-----|----|----|----|-----------|----|----|----|------------|----|----|----|--------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SWP_frac | | | | PSR_M | | | | SynchPrim_frac | | | | Barrier | | | | SMC | | | | Writeback | | | | WithShifts | | | | Unpriv | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

SWP_frac, bits [31:28]

Indicates support for the memory system locking the bus for SWP or SWPB instructions. Defined values are:

| SWP_frac | Meaning |
|----------|---|
| 0b0000 | SWP or SWPB instructions not implemented. |
| 0b0001 | SWP or SWPB implemented but only in a uniprocessor context. SWP and SWPB do not guarantee whether memory accesses from other Requesters can come between the load memory access and the store memory access of the SWP or SWPB. |

All other values are reserved. This field is valid only if [ID_ISAR0.Swap](#) is 0b0000.

In Armv8-A, the only permitted value is 0b0000.

PSR_M, bits [27:24]

Indicates the implemented M-profile instructions to modify the PSRs. Defined values are:

| PSR_M | Meaning |
|--------------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds the M-profile forms of the CPS, MRS, and MSR instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

SynchPrim_frac, bits [23:20]

Used in conjunction with [ID_ISAR3.SynchPrim](#) to indicate the implemented Synchronization Primitive instructions. Possible values are:

| SynchPrim_frac | Meaning |
|-----------------------|---|
| 0b0000 | If SynchPrim == 0b0000, no Synchronization Primitives implemented. If SynchPrim == 0b0001, adds the LDREX and STREX instructions. If SynchPrim == 0b0010, also adds the CLREX, LDREXB, LDREXH, STREXB, STREXH, LDREXD, and STREXD instructions. |
| 0b0011 | If SynchPrim == 0b0001, adds the LDREX, STREX, CLREX, LDREXB, LDREXH, STREXB, and STREXH instructions. |

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A, the only permitted value is 0b0000.

Barrier, bits [19:16]

Indicates the implemented Barrier instructions in the A32 and T32 instruction sets. Defined values are:

| Barrier | Meaning |
|----------------|---|
| 0b0000 | None implemented. Barrier operations are provided only as System instructions in the (coproc==0b1111) encoding space. |
| 0b0001 | Adds the DMB, DSB, and ISB barrier instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

SMC, bits [15:12]

Indicates the implemented SMC instructions. Defined values are:

| SMC | Meaning |
|------------|---------------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds the SMC instruction. |

All other values are reserved.

In Armv8-A, the permitted values are:

- If EL3 is implemented and EL1 can use AArch32, the only permitted value is 0b0001.
- If neither EL3 nor EL2 is implemented, the only permitted value is 0b0000.

If EL1 cannot use AArch32, this field has the value 0b0000.

Writeback, bits [11:8]

Indicates the support for Writeback addressing modes. Defined values are:

| Writeback | Meaning |
|-----------|--|
| 0b0000 | Basic support. Only the LDM, STM, PUSH, POP, SRS, and RFE instructions support writeback addressing modes. These instructions support all of their writeback addressing modes. |
| 0b0001 | Adds support for all of the writeback addressing modes. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

WithShifts, bits [7:4]

Indicates the support for instructions with shifts. Defined values are:

| WithShifts | Meaning |
|------------|--|
| 0b0000 | Nonzero shifts supported only in MOV and shift instructions. |
| 0b0001 | Adds support for shifts of loads and stores over the range LSL 0-3. |
| 0b0011 | As for 0b0001, and adds support for other constant shift options, both on load/store and other instructions. |
| 0b0100 | As for 0b0011, and adds support for register-controlled shift options. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0100.

Unpriv, bits [3:0]

Indicates the implemented unprivileged instructions. Defined values are:

| Unpriv | Meaning |
|--------|--|
| 0b0000 | None implemented. No T variant instructions are implemented. |
| 0b0001 | Adds the LDRBT, LDRT, STRBT, and STRT instructions. |
| 0b0010 | As for 0b0001, and adds the LDRHT, LDRSBT, LDRSHT, and STRHT instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR4_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_ISAR4_EL1

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b100 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_ISAR4_EL1;
    elseif PSTATE.EL == EL2 then
        return ID_ISAR4_EL1;
    elseif PSTATE.EL == EL3 then
        return ID_ISAR4_EL1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR5_EL1, AArch32 Instruction Set Attribute Register 5

The ID_ISAR5_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0_EL1](#), [ID_ISAR1_EL1](#), [ID_ISAR2_EL1](#), [ID_ISAR3_EL1](#), and [ID_ISAR4_EL1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR5_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_ISAR5\[31:0\]](#).

Attributes

ID_ISAR5_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-----|----|----|----|------|----|----|----|-------|----|----|----|------|----|----|----|------|----|----|----|-----|----|----|----|------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| VCMA | | | | RDM | | | | RES0 | | | | CRC32 | | | | SHA2 | | | | SHA1 | | | | AES | | | | SEVL | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

VCMA, bits [31:28]

Indicates AArch32 support for complex number addition and multiplication where numbers are stored in vectors. Defined values are:

| VCMA | Meaning |
|--------|--|
| 0b0000 | The VCMLA and VCADD instructions are not implemented in AArch32. |
| 0b0001 | The VCMLA and VCADD instructions are implemented in AArch32. |

All other values are reserved.

FEAT_FCMA implements the functionality identified by 0b0001.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

From Armv8.3, the only permitted value is 0b0001.

RDM, bits [27:24]

Indicates whether the VQRDMLAH and VQRDMLSH instructions are implemented in AArch32 state. Defined values are:

| RDM | Meaning |
|------------|--|
| 0b0000 | No VQRDMLAH and VQRDMLSH instructions implemented. |
| 0b0001 | VQRDMLAH and VQRDMLSH instructions implemented. |

All other values are reserved.

FEAT_RDM implements the functionality identified by the value 0b0001.

In Armv8.0, the only permitted value is 0b0000.

From Armv8.1, the only permitted value is 0b0001.

Bits [23:20]

Reserved, RES0.

CRC32, bits [19:16]

Indicates whether the CRC32 instructions are implemented in AArch32 state.

| CRC32 | Meaning |
|--------------|---|
| 0b0000 | No CRC32 instructions implemented. |
| 0b0001 | CRC32B, CRC32H, CRC32W, CRC32CB, CRC32CH, and CRC32CW instructions implemented. |

All other values are reserved.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.1, the only permitted value is 0b0001.

SHA2, bits [15:12]

Indicates whether the SHA2 instructions are implemented in AArch32 state.

| SHA2 | Meaning |
|-------------|--|
| 0b0000 | No SHA2 instructions implemented. |
| 0b0001 | SHA256H, SHA256H2, SHA256SU0, and SHA256SU1 implemented. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

SHA1, bits [11:8]

Indicates whether the SHA1 instructions are implemented in AArch32 state.

| SHA1 | Meaning |
|-------------|---|
| 0b0000 | No SHA1 instructions implemented. |
| 0b0001 | SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, and SHA1SU1 implemented. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

AES, bits [7:4]

Indicates whether the AES instructions are implemented in AArch32 state.

| AES | Meaning |
|--------|--|
| 0b0000 | No AES instructions implemented. |
| 0b0001 | AESE, AESD, AESMC, and AESIMC implemented. |
| 0b0010 | As for 0b0001, plus VMULL (polynomial) instructions operating on 64-bit data quantities. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0010.

SEVL, bits [3:0]

Indicates whether the SEVL instruction is implemented in AArch32 state.

| SEVL | Meaning |
|--------|--|
| 0b0000 | SEVL is implemented as a NOP. |
| 0b0001 | SEVL is implemented as Send Event Local. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR5_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_ISAR5_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b101 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_ISAR5_EL1;
elseif PSTATE.EL == EL2 then
    return ID_ISAR5_EL1;
elseif PSTATE.EL == EL3 then
    return ID_ISAR5_EL1;

```


ID_ISAR6_EL1, AArch32 Instruction Set Attribute Register 6

The ID_ISAR6_EL1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0_EL1](#), [ID_ISAR1_EL1](#), [ID_ISAR2_EL1](#), [ID_ISAR3_EL1](#), [ID_ISAR4_EL1](#) and [ID_ISAR5_EL1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_ISAR6_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_ISAR6\[31:0\]](#).

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_ISAR6_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|------|----|----|----|------|----|----|----|---------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|-------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | I8MM | | | | BF16 | | | | SPECRES | | | | SB | | | | FHM | | | | DP | | | | JSCVT | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:28]

Reserved, RES0.

I8MM, bits [27:24]

Indicates support for Advanced SIMD and floating-point Int8 matrix multiplication instructions in AArch32 state. Defined values of this field are:

| I8MM | Meaning |
|--------|---|
| 0b0000 | Int8 matrix multiplication instructions are not implemented. |
| 0b0001 | VSMMLA, VSUDOT, VUMMLA, VUSMMLA, and VUSDOT instructions are implemented. |

All other values are reserved.

FEAT_AA32I8MM implements the functionality identified by 0b0001.

BF16, bits [23:20]

Indicates support for Advanced SIMD and floating-point BFloat16 instructions in AArch32 state. Defined values are:

| BF16 | Meaning |
|-------------|--|
| 0b0000 | BFloat16 instructions are not implemented. |
| 0b0001 | VCVT, VCVTB, VCVTT, VDOT, VFMA, VFMA, and VMLA instructions with BF16 operand or result types are implemented. |

All other values are reserved.

FEAT_AA32BF16 implements the functionality identified by 0b0001.

SPECRES, bits [19:16]

Indicates support for Speculation invalidation instructions in AArch32 state. Defined values are:

| SPECRES | Meaning |
|----------------|---|
| 0b0000 | Prediction invalidation instructions are not implemented. |
| 0b0001 | CFPRCTX, DVPRCTX, and CPPRCTX instructions are implemented. |

All other values are reserved.

FEAT_SPECRES implements the functionality identified by 0b0001.

From Armv8.5, the only permitted value is 0b0001.

SB, bits [15:12]

Indicates support for the SB instruction in AArch32 state. Defined values are:

| SB | Meaning |
|-----------|------------------------------------|
| 0b0000 | SB instruction is not implemented. |
| 0b0001 | SB instruction is implemented. |

All other values are reserved.

FEAT_SB implements the functionality identified by 0b0001.

From Armv8.5, the only permitted value is 0b0001.

FHM, bits [11:8]

Indicates support for Advanced SIMD and floating-point VFMA and VFMSL instructions in AArch32 state. Defined values are:

| FHM | Meaning |
|------------|--|
| 0b0000 | VFMA and VFMSL instructions are not implemented. |
| 0b0001 | VFMA and VFMSL instructions are implemented. |

All other values are reserved.

FEAT_FHM implements the functionality identified by 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

DP, bits [7:4]

Indicates support for dot product instructions in AArch32 state. Defined values are:

| DP | Meaning |
|-----------|---|
| 0b0000 | Dot product instructions are not implemented. |
| 0b0001 | VUDOT and VSDOT instructions are implemented. |

All other values are reserved.

FEAT_DotProd implements the functionality identified by 0b0001.

In Armv8.2, the permitted values are 0b0000 and 0b0001.

From Armv8.4, the only permitted value is 0b0001.

JSCVT, bits [3:0]

Indicates support for the VJCVT instruction in AArch32 state. Defined values are:

| JSCVT | Meaning |
|--------|---|
| 0b0000 | The VJCVT instruction is not implemented. |
| 0b0001 | The VJCVT instruction is implemented. |

All other values are reserved.

FEAT_JSCVT implements the functionality identified by 0b0001.

In Armv8.0, Armv8.1, and Armv8.2, the only permitted value is 0b0000.

From Armv8.3, if Advanced SIMD or Floating-point is implemented, the only permitted value is 0b0001.

From Armv8.3, if Advanced SIMD or Floating-point is not implemented, the only permitted value is 0b0000.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_ISAR6_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_ISAR6_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b111 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && (!IsZero(ID_ISAR6_EL1) || boolean IMPLEMENTATION_DEFINED "ID_ISAR6_EL1
trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_ISAR6_EL1;
elseif PSTATE.EL == EL2 then
    return ID_ISAR6_EL1;
elseif PSTATE.EL == EL3 then
    return ID_ISAR6_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR0_EL1, AArch32 Memory Model Feature Register 0

The ID_MMFR0_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_MMFR0\[31:0\]](#).

Attributes

ID_MMFR0_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|------|----|----|----|--------|----|----|----|-----|----|----|----|----------|----|----|----|----------|----|----|----|------|----|----|----|------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| InnerShr | | | | FCSE | | | | AuxReg | | | | TCM | | | | ShareLvl | | | | OuterShr | | | | PMSA | | | | VMSA | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

InnerShr, bits [31:28]

Innermost Shareability. Indicates the innermost shareability domain implemented. Defined values are:

| InnerShr | Meaning |
|----------|--|
| 0b0000 | Implemented as Non-cacheable. |
| 0b0001 | Implemented with hardware coherency support. |
| 0b1111 | Shareability ignored. |

All other values are reserved.

From Armv8 the permitted values are 0b0000, 0b0001, and 0b1111.

This field is valid only if the implementation supports two levels of shareability, as indicated by ID_MMFR0_EL1.ShareLvl having the value 0b0001.

When ID_MMFR0_EL1.ShareLvl is zero, this field is UNKNOWN.

FCSE, bits [27:24]

Indicates whether the implementation includes the FCSE. Defined values are:

| FCSE | Meaning |
|--------|-------------------|
| 0b0000 | Not supported. |
| 0b0001 | Support for FCSE. |

All other values are reserved.

From Armv8 the only permitted value is 0b0000.

AuxReg, bits [23:20]

Auxiliary Registers. Indicates support for Auxiliary registers. Defined values are:

| AuxReg | Meaning |
|--------|--|
| 0b0000 | None supported. |
| 0b0001 | Support for Auxiliary Control Register only. |
| 0b0010 | Support for Auxiliary Fault Status Registers (AIFSR and ADFSR) and Auxiliary Control Register. |

All other values are reserved.

From Armv8 the only permitted value is 0b0010.

Note

Accesses to unimplemented Auxiliary registers are UNDEFINED.

TCM, bits [19:16]

Indicates support for TCMs and associated DMAs. Defined values are:

| TCM | Meaning |
|--------|---|
| 0b0000 | Not supported. |
| 0b0001 | Support is IMPLEMENTATION DEFINED. Armv7 requires this setting. |
| 0b0010 | Support for TCM only, Armv6 implementation. |
| 0b0011 | Support for TCM and DMA, Armv6 implementation. |

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

ShareLvl, bits [15:12]

Shareability Levels. Indicates the number of shareability levels implemented. Defined values are:

| ShareLvl | Meaning |
|----------|---|
| 0b0000 | One level of shareability implemented. |
| 0b0001 | Two levels of shareability implemented. |

All other values are reserved.

From Armv8 the only permitted value is 0b0001.

OuterShr, bits [11:8]

Outermost Shareability. Indicates the outermost shareability domain implemented. Defined values are:

| OuterShr | Meaning |
|----------|--|
| 0b0000 | Implemented as Non-cacheable. |
| 0b0001 | Implemented with hardware coherency support. |
| 0b1111 | Shareability ignored. |

All other values are reserved.

From Armv8 the permitted values are 0b0000, 0b0001, and 0b1111.

PMSA, bits [7:4]

Indicates support for a PMSA. Defined values are:

| PMSA | Meaning |
|--------|---|
| 0b0000 | Not supported. |
| 0b0001 | Support for IMPLEMENTATION DEFINED PMSA. |
| 0b0010 | Support for PMSAv6, with a Cache Type Register implemented. |
| 0b0011 | Support for PMSAv7, with support for memory subsections. Armv7-R profile. |

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

VMSA, bits [3:0]

Indicates support for a VMSA. Defined values are:

| VMSA | Meaning |
|--------|---|
| 0b0000 | Not supported. |
| 0b0001 | Support for IMPLEMENTATION DEFINED VMSA. |
| 0b0010 | Support for VMSAv6, with Cache and TLB Type Registers implemented. |
| 0b0011 | Support for VMSAv7, with support for remapping and the Access flag. Armv7-A profile. |
| 0b0100 | As for 0b0011, and adds support for the PXN bit in the Short-descriptor translation table format descriptors. |
| 0b0101 | As for 0b0100, and adds support for the Long-descriptor translation table format. |

All other values are reserved.

In Armv8-A the only permitted value is 0b0101.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_MMFR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b100 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_MMFR0_EL1;
elseif PSTATE.EL == EL2 then
    return ID_MMFR0_EL1;
elseif PSTATE.EL == EL3 then
    return ID_MMFR0_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR1_EL1, AArch32 Memory Model Feature Register 1

The ID_MMFR1_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_MMFR1\[31:0\]](#).

Attributes

ID_MMFR1_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----------|----|----|----|-------|----|----|----|-------|----|----|----|---------|----|----|----|---------|----|----|----|---------|----|----|----|---------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| BPred | | | | L1TstCln | | | | L1Uni | | | | L1Hvd | | | | L1UniSW | | | | L1HvdSW | | | | L1UniVA | | | | L1HvdVA | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

BPred, bits [31:28]

Branch Predictor. Indicates branch predictor management requirements. Defined values are:

| BPred | Meaning |
|--------------|--|
| 0b0000 | No branch predictor, or no MMU present. Implies a fixed MPU configuration. |
| 0b0001 | Branch predictor requires flushing on: <ul style="list-style-type: none"> • Enabling or disabling a stage of address translation. • Writing new data to instruction locations. • Writing new mappings to the translation tables. • Changes to the TTBR0, TTBR1, or TTBCR registers. • Changes to the ContextID or ASID, or to the FCSE ProcessID if this is supported. |
| 0b0010 | Branch predictor requires flushing on: <ul style="list-style-type: none"> • Enabling or disabling a stage of address translation. • Writing new data to instruction locations. • Writing new mappings to the translation tables. • Any change to the TTBR0, TTBR1, or TTBCR registers without a change to the corresponding ContextID or ASID, or FCSE ProcessID if this is supported. |
| 0b0011 | Branch predictor requires flushing only on writing new data to instruction locations. |
| 0b0100 | For execution correctness, branch predictor requires no flushing at any time. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0010, 0b0011, and 0b0100. For values other than 0b0000 and 0b0100 the Arm Architecture Reference Manual, or the product documentation, might give more information about the required maintenance.

L1TstCln, bits [27:24]

Level 1 cache Test and Clean. Indicates the supported Level 1 data cache test and clean operations, for Harvard or unified cache implementations. Defined values are:

| L1TstCln | Meaning |
|-----------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported Level 1 data cache test and clean operations are: <ul style="list-style-type: none"> • Test and clean data cache. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Test, clean, and invalidate data cache. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1Uni, bits [23:20]

Level 1 Unified cache. Indicates the supported entire Level 1 cache maintenance operations for a unified cache implementation. Defined values are:

| L1Uni | Meaning |
|--------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported entire Level 1 cache operations are: <ul style="list-style-type: none"> • Invalidate cache, including branch predictor if appropriate. • Invalidate branch predictor, if appropriate. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Clean cache, using a recursive model that uses the cache dirty status bit. • Clean and invalidate cache, using a recursive model that uses the cache dirty status bit. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1Hvd, bits [19:16]

Level 1 Harvard cache. Indicates the supported entire Level 1 cache maintenance operations for a Harvard cache implementation. Defined values are:

| L1Hvd | Meaning |
|--------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported entire Level 1 cache operations are: <ul style="list-style-type: none"> • Invalidate instruction cache, including branch predictor if appropriate. • Invalidate branch predictor, if appropriate. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Invalidate data cache. • Invalidate data cache and instruction cache, including branch predictor if appropriate. |
| 0b0011 | As for 0b0010, and adds: <ul style="list-style-type: none"> • Clean data cache, using a recursive model that uses the cache dirty status bit. • Clean and invalidate data cache, using a recursive model that uses the cache dirty status bit. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1UniSW, bits [15:12]

Level 1 Unified cache by Set/Way. Indicates the supported Level 1 cache line maintenance operations by set/way, for a unified cache implementation. Defined values are:

| L1UniSW | Meaning |
|----------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported Level 1 unified cache line maintenance operations by set/way are: <ul style="list-style-type: none"> • Clean cache line by set/way. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Clean and invalidate cache line by set/way. |
| 0b0011 | As for 0b0010, and adds: <ul style="list-style-type: none"> • Invalidate cache line by set/way. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1HvdSW, bits [11:8]

Level 1 Harvard cache by Set/Way. Indicates the supported Level 1 cache line maintenance operations by set/way, for a Harvard cache implementation. Defined values are:

| L1HvdSW | Meaning |
|----------------|---|
| 0b0000 | None supported. |
| 0b0001 | Supported Level 1 Harvard cache line maintenance operations by set/way are: <ul style="list-style-type: none"> • Clean data cache line by set/way. • Clean and invalidate data cache line by set/way. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Invalidate data cache line by set/way. |
| 0b0011 | As for 0b0010, and adds: <ul style="list-style-type: none"> • Invalidate instruction cache line by set/way. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1UniVA, bits [7:4]

Level 1 Unified cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a unified cache implementation. Defined values are:

| L1UniVA | Meaning |
|----------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported Level 1 unified cache line maintenance operations by VA are: <ul style="list-style-type: none"> • Clean cache line by VA. • Invalidate cache line by VA. • Clean and invalidate cache line by VA. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Invalidate branch predictor by VA, if branch predictor is implemented. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1HvdVA, bits [3:0]

Level 1 Harvard cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a Harvard cache implementation. Defined values are:

| L1HvdVA | Meaning |
|----------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported Level 1 Harvard cache line maintenance operations by VA are: <ul style="list-style-type: none"> • Clean data cache line by VA. • Invalidate data cache line by VA. • Clean and invalidate data cache line by VA. • Clean instruction cache line by VA. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Invalidate branch predictor by VA, if branch predictor is implemented. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_MMFR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b101 |

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_MMFR1_EL1;
elseif PSTATE.EL == EL2 then
    return ID_MMFR1_EL1;
elseif PSTATE.EL == EL3 then
    return ID_MMFR1_EL1;
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR2_EL1, AArch32 Memory Model Feature Register 2

The ID_MMFR2_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_MMFR2\[31:0\]](#).

Attributes

ID_MMFR2_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----------|----|----|----|---------|----|----|----|--------|----|----|----|--------|----|----|----|----------|----|----|----|---------|----|----|----|---------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| HWAccFlg | | | | WFIStall | | | | MemBarr | | | | UniTLB | | | | HvdTLB | | | | L1HvdRng | | | | L1HvdBG | | | | L1HvdFG | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

HWAccFlg, bits [31:28]

Hardware Access Flag. In earlier versions of the Arm Architecture, this field indicates support for a Hardware Access flag, as part of the VMSAv7 implementation. Defined values are:

| HWAccFlg | Meaning |
|----------|--|
| 0b0000 | Not supported. |
| 0b0001 | Support for VMSAv7 Access flag, updated in hardware. |

All other values are reserved.

From Armv8, the only permitted value is 0b0000.

WFIStall, bits [27:24]

Wait For Interrupt Stall. Indicates the support for Wait For Interrupt (WFI) stalling. Defined values are:

| WFIStall | Meaning |
|----------|---------------------------|
| 0b0000 | Not supported. |
| 0b0001 | Support for WFI stalling. |

All other values are reserved.

From Armv8, the permitted values are 0b0000 and 0b0001.

MemBarr, bits [23:20]

Memory Barrier. Indicates the supported memory barrier System instructions in the (coproc==0b1111) encoding space:

| MemBarr | Meaning |
|---------|---|
| 0b0000 | None supported. |
| 0b0001 | Supported memory barrier System instructions are: <ul style="list-style-type: none"> • Data Synchronization Barrier (DSB). |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Instruction Synchronization Barrier (ISB). • Data Memory Barrier (DMB). |

All other values are reserved.

From Armv8, the only permitted value is 0b0010.

Arm deprecates the use of these operations. ID_ISAR4.Barrier_instrs indicates the level of support for the preferred barrier instructions.

UniTLB, bits [19:16]

Unified TLB. Indicates the supported TLB maintenance operations, for a unified TLB implementation. Defined values are:

| UniTLB | Meaning |
|--------|---|
| 0b0000 | Not supported. |
| 0b0001 | Supported unified TLB maintenance operations are: <ul style="list-style-type: none"> • Invalidate all entries in the TLB. • Invalidate TLB entry by VA. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Invalidate TLB entries by ASID match. |
| 0b0011 | As for 0b0010, and adds: <ul style="list-style-type: none"> • Invalidate instruction TLB and data TLB entries by VA All ASID. This is a shared unified TLB operation. |
| 0b0100 | As for 0b0011, and adds: <ul style="list-style-type: none"> • Invalidate Hyp mode unified TLB entry by VA. • Invalidate entire Non-secure PL1&0 unified TLB. • Invalidate entire Hyp mode unified TLB. |
| 0b0101 | As for 0b0100, and adds the following operations: TLBIMVALIS , TLBIMVAALIS , TLBIMVALHIS , TLBIMVAL , TLBIMVAAL , TLBIMVALH . |
| 0b0110 | As for 0b0101, and adds the following operations: TLBIIPAS2IS , TLBIIPAS2LIS , TLBIIPAS2 , TLBIIPAS2L . |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0110.

HvdTLB, bits [15:12]

If the Unified TLB field (UniTLB, bits [19:16]) is not 0000, then the meaning of this field is IMPLEMENTATION DEFINED. Arm deprecates the use of this field by software.

L1HvdRng, bits [11:8]

Level 1 Harvard cache Range. Indicates the supported Level 1 cache maintenance range operations, for a Harvard cache implementation. Defined values are:

| L1HvdRng | Meaning |
|-----------------|--|
| 0b0000 | Not supported. |
| 0b0001 | Supported Level 1 Harvard cache maintenance range operations are: <ul style="list-style-type: none"> • Invalidate data cache range by VA. • Invalidate instruction cache range by VA. • Clean data cache range by VA. • Clean and invalidate data cache range by VA. |

All other values are reserved.

From Armv8, the only permitted value is 0b0000.

L1HvdBG, bits [7:4]

Level 1 Harvard cache Background fetch. Indicates the supported Level 1 cache background fetch operations, for a Harvard cache implementation. When supported, background fetch operations are non-blocking operations. Defined values are:

| L1HvdBG | Meaning |
|----------------|--|
| 0b0000 | Not supported. |
| 0b0001 | Supported Level 1 Harvard cache background fetch operations are: <ul style="list-style-type: none"> • Fetch instruction cache range by VA. • Fetch data cache range by VA. |

All other values are reserved.

From Armv8, the only permitted value is 0b0000.

L1HvdFG, bits [3:0]

Level 1 Harvard cache Foreground fetch. Indicates the supported Level 1 cache foreground fetch operations, for a Harvard cache implementation. When supported, foreground fetch operations are blocking operations. Defined values are:

| L1HvdFG | Meaning |
|----------------|--|
| 0b0000 | Not supported. |
| 0b0001 | Supported Level 1 Harvard cache foreground fetch operations are: <ul style="list-style-type: none"> • Fetch instruction cache range by VA. • Fetch data cache range by VA. |

All other values are reserved.

From Armv8, the only permitted value is 0b0000.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_MMFR2_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b110 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_MMFR2_EL1;
    elsif PSTATE.EL == EL2 then
        return ID_MMFR2_EL1;
    elsif PSTATE.EL == EL3 then
        return ID_MMFR2_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR3_EL1, AArch32 Memory Model Feature Register 3

The ID_MMFR3_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR3_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_MMFR3\[31:0\]](#).

Attributes

ID_MMFR3_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|--------|----|----|----|---------|----|----|----|-----|----|----|----|-----------|----|----|----|---------|----|----|----|----------|----|----|----|----------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Supersec | | | | CMemSz | | | | CohWalk | | | | PAN | | | | MaintBcst | | | | BPMaint | | | | CMaintSW | | | | CMaintVA | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

Supersec, bits [31:28]

Supersections. On a VMSA implementation, indicates whether Supersections are supported. Defined values are:

| Supersec | Meaning |
|----------|------------------------------|
| 0b0000 | Supersections supported. |
| 0b1111 | Supersections not supported. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b1111.

CMemSz, bits [27:24]

Cached Memory Size. Indicates the physical memory size supported by the caches. Defined values are:

| CMemSz | Meaning |
|--------|--|
| 0b0000 | 4GB, corresponding to a 32-bit physical address range. |
| 0b0001 | 64GB, corresponding to a 36-bit physical address range. |
| 0b0010 | 1TB or more, corresponding to a 40-bit or larger physical address range. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000, 0b0001, and 0b0010.

CohWalk, bits [23:20]

Coherent Walk. Indicates whether Translation table updates require a clean to the Point of Unification. Defined values are:

| CohWalk | Meaning |
|---------|--|
| 0b0000 | Updates to the translation tables require a clean to the Point of Unification to ensure visibility by subsequent translation table walks. |
| 0b0001 | Updates to the translation tables do not require a clean to the Point of Unification to ensure visibility by subsequent translation table walks. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

PAN, bits [19:16]

Privileged Access Never. Indicates support for the PAN bit in [CPSR](#), [SPSR](#), and [DPSR](#) in AArch32 state. Defined values are:

| PAN | Meaning |
|--------|---|
| 0b0000 | PAN not supported. |
| 0b0001 | PAN supported. |
| 0b0010 | PAN supported and ATS1CPRP and ATS1CPWP instructions supported. |

All other values are reserved.

FEAT_PAN implements the functionality identified by the value 0b0001.

FEAT_PAN2 implements the functionality added by the value 0b0010.

In Armv8.1, the value 0b0000 is not permitted.

From Armv8.2, the only permitted value is 0b0010.

MaintBcst, bits [15:12]

Maintenance Broadcast. Indicates whether Cache, TLB, and branch predictor operations are broadcast. Defined values are:

| MaintBcst | Meaning |
|-----------|--|
| 0b0000 | Cache, TLB, and branch predictor operations only affect local structures. |
| 0b0001 | Cache and branch predictor operations affect structures according to shareability and defined behavior of instructions. TLB operations only affect local structures. |
| 0b0010 | Cache, TLB, and branch predictor operations affect structures according to shareability and defined behavior of instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

BPMaint, bits [11:8]

Branch Predictor Maintenance. Indicates the supported branch predictor maintenance operations in an implementation with hierarchical cache maintenance operations. Defined values are:

| BPMaint | Meaning |
|----------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported branch predictor maintenance operations are: <ul style="list-style-type: none"> • Invalidate all branch predictors. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Invalidate branch predictors by VA. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

CMaintSW, bits [7:4]

Cache Maintenance by Set/Way. Indicates the supported cache maintenance operations by set/way, in an implementation with hierarchical caches. Defined values are:

| CMaintSW | Meaning |
|-----------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported hierarchical cache maintenance instructions by set/way are: <ul style="list-style-type: none"> • Invalidate data cache by set/way. • Clean data cache by set/way. • Clean and invalidate data cache by set/way. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

In a unified cache implementation, the data cache maintenance operations apply to the unified caches.

CMaintVA, bits [3:0]

Cache Maintenance by Virtual Address. Indicates the supported cache maintenance operations by VA, in an implementation with hierarchical caches. Defined values are:

| CMaintVA | Meaning |
|-----------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported hierarchical cache maintenance operations by VA are: <ul style="list-style-type: none"> • Invalidate data cache by VA. • Clean data cache by VA. • Clean and invalidate data cache by VA. • Invalidate instruction cache by VA. • Invalidate all instruction cache entries. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

In a unified cache implementation, data cache maintenance operations apply to the unified caches, and the instruction cache maintenance instructions are not implemented.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR3_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_MMFR3_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b111 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_MMFR3_EL1;
elseif PSTATE.EL == EL2 then
    return ID_MMFR3_EL1;
elseif PSTATE.EL == EL3 then
    return ID_MMFR3_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR4_EL1, AArch32 Memory Model Feature Register 4

The ID_MMFR4_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR4_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_MMFR4\[31:0\]](#).

Attributes

ID_MMFR4_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|----|----|-----|----|----|----|------|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|---------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| EVT | | | | CCIDX | | | | LSM | | | | HPDS | | | | CnP | | | | XNX | | | | AC2 | | | | SpecSEI | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

EVT, bits [31:28]

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the [HCR2](#).{TTLBIS, TOCU, TICAB, TID4} traps. Defined values are:

| EVT | Meaning |
|--------|---|
| 0b0000 | HCR2 .{TTLBIS, TOCU, TICAB, TID4} traps are not supported. |
| 0b0001 | HCR2 .{TOCU, TICAB, TID4} traps are supported. HCR2 .TTLBIS trap is not supported. |
| 0b0010 | HCR2 .{TTLBIS, TOCU, TICAB, TID4} traps are supported. |

All other values are reserved.

FEAT_EVT implements the functionality identified by the values 0b0001 and 0b0010.

If EL2 is not implemented supporting AArch32, the only permitted value is 0b0000.

In Armv8.2, the permitted values are 0b0000, 0b0001, and 0b0010.

From Armv8.5, the permitted values are:

- 0b0000 when EL2 is not implemented or does not support AArch32.
- 0b0010 when EL2 is implemented and supports AArch32.

CCIDX, bits [27:24]

Support for use of the revised CCSIDR format and the presence of the CCSIDR2 is indicated. Defined values are:

| CCIDX | Meaning |
|--------|--|
| 0b0000 | 32-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is not implemented. |
| 0b0001 | 64-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is implemented. |

All other values are reserved.

FEAT_CCIDX implements the functionality identified by 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

LSM, bits [23:20]

Indicates support for LSMAOE and nTLSMD bits in [HSCTLR](#) and [SCTLR](#). Defined values are:

| LSM | Meaning |
|--------|---------------------------------------|
| 0b0000 | LSMAOE and nTLSMD bits not supported. |
| 0b0001 | LSMAOE and nTLSMD bits supported. |

All other values are reserved.

FEAT_LSMAOC implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

HPDS, bits [19:16]

Hierarchical permission disables bits in translation tables. Defined values are:

| HPDS | Meaning |
|--------|--|
| 0b0000 | Disabling of hierarchical controls not supported. |
| 0b0001 | Supports disabling of hierarchical controls using the TTBCR2 .HPD0, TTBCR2 .HPD1, and HTCR .HPD bits. |
| 0b0010 | As for value 0b0001, and adds possible hardware allocation of bits[62:59] of the translation table descriptors from the final lookup level for IMPLEMENTATION DEFINED use. |

All other values are reserved.

FEAT_AA32HPD implements the functionality identified by the value 0b0001.

FEAT_HPDS2 implements the functionality added by the value 0b0010.

Note

The value 0b0000 implies that the encoding for [TTBCR2](#) is UNDEFINED.

CnP, bits [15:12]

Common not Private translations. Defined values are:

| CnP | Meaning |
|--------|--|
| 0b0000 | Common not Private translations not supported. |
| 0b0001 | Common not Private translations supported. |

All other values are reserved.

FEAT_TTCNP implements the functionality identified by the value 0b0001.

From Armv8.2 the only permitted value is 0b0001.

XNX, bits [11:8]

Support for execute-never control distinction by Exception level at stage 2. Defined values are:

| XNX | Meaning |
|--------|---|
| 0b0000 | Distinction between EL0 and EL1 execute-never control at stage 2 not supported. |
| 0b0001 | Distinction between EL0 and EL1 execute-never control at stage 2 supported. |

All other values are reserved.

FEAT_XNX implements the functionality identified by the value 0b0001.

When FEAT_XNX is implemented:

- If all of the following conditions are true, it is IMPLEMENTATION DEFINED whether the value of ID_MMFR4_EL1.XNX is 0b0000 or 0b0001:
 - ID_AA64MMFR1_EL1.XNX == 1.
 - EL2 cannot use AArch32.
 - EL1 can use AArch32.
- If EL2 can use AArch32 then the only permitted value is 0b0001.

AC2, bits [7:4]

Indicates the extension of the [ACTLR](#) and [HACTLR](#) registers using [ACTLR2](#) and [HACTLR2](#). Defined values are:

| AC2 | Meaning |
|--------|---|
| 0b0000 | ACTLR2 and HACTLR2 are not implemented. |
| 0b0001 | ACTLR2 and HACTLR2 are implemented. |

All other values are reserved.

In Armv8.0 and Armv8.1 the permitted values are 0b0000 and 0b0001.

From Armv8.2, the only permitted value is 0b0001.

SpecSEI, bits [3:0]

Describes whether the PE can generate SError interrupt exceptions from speculative reads of memory, including speculative instruction fetches. The defined values of this field are:

| SpecSEI | Meaning |
|---------|--|
| 0b0000 | The PE never generates an SError interrupt due to an External abort on a speculative read. |
| 0b0001 | The PE might generate an SError interrupt due to an External abort on a speculative read. |

All other values are reserved.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR4_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_MMFR4_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && (!IsZero(ID_MMFR4_EL1) || boolean IMPLEMENTATION_DEFINED "ID_MMFR4_EL1
trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_MMFR4_EL1;
    elsif PSTATE.EL == EL2 then
        return ID_MMFR4_EL1;
    elsif PSTATE.EL == EL3 then
        return ID_MMFR4_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR5_EL1, AArch32 Memory Model Feature Register 5

The ID_MMFR5_EL1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_MMFR5_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_MMFR5\[31:0\]](#).

Attributes

ID_MMFR5_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | nTLBPA | | | | ETS | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:8]

Reserved, RES0.

nTLBPA, bits [7:4]

Indicates support for intermediate caching of translation table walks. Defined values are:

| nTLBPA | Meaning |
|--------|---|
| 0b0000 | The intermediate caching of translation table walks might include non-coherent caches of previous valid translation table entries since the last completed relevant TLBI applicable to the PE where either: <ul style="list-style-type: none">The caching is indexed by the physical address of the location holding the translation table entry.The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry. |
| 0b0001 | The intermediate caching of translation table walks does not include non-coherent caches of previous valid translation table entries since the last completed TLBI applicable to the PE where either: <ul style="list-style-type: none">The caching is indexed by the physical address of the location holding the translation table entry.The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry. |

All other values are reserved.

FEAT_nTLBPA implements the functionality identified by the value 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

ETS, bits [3:0]

Indicates support for Enhanced Translation Synchronization. Defined values are:

| ETS | Meaning |
|--------|--|
| 0b0000 | Enhanced Translation Synchronization is not supported. |
| 0b0001 | Enhanced Translation Synchronization is supported. |

All other values are reserved.

FEAT_ETTS implements the functionality identified by the value 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.7, the only permitted value is 0b0001.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_MMFR5_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_MMFR5_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0011 | 0b110 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && (!IsZero(ID_MMFR5_EL1) || boolean IMPLEMENTATION_DEFINED "ID_MMFR5_EL1
trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_MMFR5_EL1;
    elsif PSTATE.EL == EL2 then
        return ID_MMFR5_EL1;
    elsif PSTATE.EL == EL3 then
        return ID_MMFR5_EL1;

```


ID_PFR0_EL1, AArch32 Processor Feature Register 0

The ID_PFR0_EL1 characteristics are:

Purpose

Gives top-level information about the instruction sets supported by the PE in AArch32 state.

Must be interpreted with [ID_PFR1_EL1](#).

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_PFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_PFR0\[31:0\]](#).

Attributes

ID_PFR0_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-----|----|----|----|-----|----|----|----|------|----|----|----|--------|----|----|----|--------|----|----|----|--------|----|----|----|--------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RAS | | | | DIT | | | | AMU | | | | CSV2 | | | | State3 | | | | State2 | | | | State1 | | | | State0 | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

RAS, bits [31:28]

RAS Extension version. Defined values are:

| RAS | Meaning |
|--------|---|
| 0b0000 | No RAS Extension. |
| 0b0001 | RAS Extension implemented. |
| 0b0010 | FEAT_RASv1p1 implemented. As 0b0001, and adds support for additional ERXMISC<m> System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS and support for the optional RAS Timestamp Extension. |

All other values are reserved.

FEAT_RAS implements the functionality identified by the value 0b0001.

FEAT_RASv1p1 implements the functionality identified by the value 0b0010.

In Armv8.0 and Armv8.1, the permitted values are 0b0000 and 0b0001.

In Armv8.2, the only permitted value is 0b0001.

From Armv8.4, if FEAT_DoubleFault is implemented, the only permitted value is 0b0010.

From Armv8.4, when FEAT_DoubleFault is not implemented, and [ERRIDR_EL1.NUM](#) is 0, the permitted values are IMPLEMENTATION DEFINED 0b0001 or 0b0010.

Note

When the value of this field is 0b0001, [ID_PFR2_EL1.RAS_frac](#) indicates whether FEAT_RASv1p1 is implemented.

DIT, bits [27:24]

Data Independent Timing. Defined values are:

| DIT | Meaning |
|--------|---|
| 0b0000 | AArch32 does not guarantee constant execution time of any instructions. |
| 0b0001 | AArch32 provides the PSTATE.DIT mechanism to guarantee constant execution time of certain instructions. |

All other values are reserved.

FEAT_DIT implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0001.

AMU, bits [23:20]

Indicates support for Activity Monitors Extension. Defined values are:

| AMU | Meaning |
|--------|--|
| 0b0000 | Activity Monitors Extension is not implemented. |
| 0b0001 | FEAT_AMUv1 is implemented. |
| 0b0010 | FEAT_AMUv1p1 is implemented. As 0b0001 and adds support for virtualization of the activity monitor event counters. |

All other values are reserved.

FEAT_AMUv1 implements the functionality identified by the value 0b0001.

FEAT_AMUv1p1 implements the functionality identified by the value 0b0010.

In Armv8.0, the only permitted value is 0b0000.

In Armv8.4, the permitted values are 0b0000 and 0b0001.

From Armv8.6, the permitted values are 0b0000, 0b0001, and 0b0010.

CSV2, bits [19:16]

Speculative use of out of context branch targets. Defined values are:

| CSV2 | Meaning |
|--------|--|
| 0b0000 | This PE does not disclose whether branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context. |
| 0b0001 | Branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context only in a hard-to-determine way. |
| 0b0010 | Branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context only in a hard-to-determine way. Within a hardware-described context, branch targets trained for branches situated at one address can control speculative execution of branches situated at different addresses only in a hard-to-determine way. |

All other values are reserved.

FEAT_CSV2 implements the functionality identified by the values 0b0001 and 0b0010.

From Armv8.5, the permitted values are 0b0001 and 0b0010.

State3, bits [15:12]

T32EE instruction set support. Defined values are:

| State3 | Meaning |
|--------|------------------------------------|
| 0b0000 | Not implemented. |
| 0b0001 | T32EE instruction set implemented. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

State2, bits [11:8]

Jazelle extension support. Defined values are:

| State2 | Meaning |
|--------|---|
| 0b0000 | Not implemented. |
| 0b0001 | Jazelle extension implemented, without clearing of JOSCR.CV on exception entry. |
| 0b0010 | Jazelle extension implemented, with clearing of JOSCR.CV on exception entry. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

State1, bits [7:4]

T32 instruction set support. Defined values are:

| State1 | Meaning |
|--------|--|
| 0b0000 | T32 instruction set not implemented. |
| 0b0001 | T32 encodings before the introduction of Thumb-2 technology implemented: <ul style="list-style-type: none"> • All instructions are 16-bit. • A BL or BLX is a pair of 16-bit instructions. • 32-bit instructions other than BL and BLX cannot be encoded. |
| 0b0011 | T32 encodings after the introduction of Thumb-2 technology implemented, for all 16-bit and 32-bit T32 basic instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

State0, bits [3:0]

A32 instruction set support. Defined values are:

| State0 | Meaning |
|--------|--------------------------------------|
| 0b0000 | A32 instruction set not implemented. |
| 0b0001 | A32 instruction set implemented. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_PFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_PFR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return ID_PFR0_EL1;
    elsif PSTATE.EL == EL2 then
        return ID_PFR0_EL1;
    elsif PSTATE.EL == EL3 then
        return ID_PFR0_EL1;

```

ID_PFR1_EL1, AArch32 Processor Feature Register 1

The ID_PFR1_EL1 characteristics are:

Purpose

Gives information about the AArch32 programmers' model.

Must be interpreted with [ID_PFR0_EL1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_PFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_PFR1\[31:0\]](#).

Attributes

ID_PFR1_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-----------|----|----|----|----------|----|----|----|----------|----|----|----|----------------|----|----|----|----------|----|----|----|----------|----|----|----|---------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| GIC | | | | Virt_frac | | | | Sec_frac | | | | GenTimer | | | | Virtualization | | | | MProgMod | | | | Security | | | | ProgMod | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

GIC, bits [31:28]

System register GIC CPU interface. Defined values are:

| GIC | Meaning |
|--------|--|
| 0b0000 | GIC CPU interface system registers not implemented. |
| 0b0001 | System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported. |
| 0b0011 | System register interface to version 4.1 of the GIC CPU interface is supported. |

All other values are reserved.

Virt_frac, bits [27:24]

Virtualization fractional field. When the Virtualization field is 0b0000, determines the support for Virtualization Extensions. Defined values are:

| Virt_frac | Meaning |
|------------------|--|
| 0b0000 | No Virtualization Extensions are implemented. |
| 0b0001 | The following Virtualization Extensions are implemented: <ul style="list-style-type: none"> • The SCR.SIF bit, if EL3 is implemented. • The modifications to the SCR.AW and SCR.FW bits described in the Virtualization Extensions, if EL3 is implemented. • The MSR (banked register) and MRS (banked register) instructions. • The ERET instruction. |

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL2 is implemented.
- 0b0001 when EL2 is not implemented.

This field is valid only when the value of ID_PFR1_EL1.Virtualization is 0, otherwise it holds the value 0b0000.

Note

The ID_ISAR registers do not identify whether the instructions added by the Virtualization Extensions are implemented.

Sec_frac, bits [23:20]

Security fractional field. When the Security field is 0b0000, determines the support for Security Extensions. Defined values are:

| Sec_frac | Meaning |
|-----------------|--|
| 0b0000 | No Security Extensions are implemented. |
| 0b0001 | The following Security Extensions are implemented: <ul style="list-style-type: none"> • The VBAR register. • The TTBCR.PD0 and TTBCR.PD1 bits. |
| 0b0010 | As for 0b0001, plus the ability to access Secure or Non-secure physical memory is supported. |

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL3 is implemented.
- 0b0001 or 0b0010 when EL3 is not implemented.

This field is valid only when the value of ID_PFR1_EL1.Security is 0, otherwise it holds the value 0b0000.

GenTimer, bits [19:16]

Generic Timer support. Defined values are:

| GenTimer | Meaning |
|-----------------|--|
| 0b0000 | Generic Timer is not implemented. |
| 0b0001 | Generic Timer is implemented. |
| 0b0010 | Generic Timer is implemented, and also includes support for CNTHCTL .EVNTIS and CNTKCTL .EVNTIS fields, and CNTPCTSS and CNTVCTSS counter views. |

All other values are reserved.

FEAT_ECV implements the functionality identified by the value 0b0010.

In Armv8.0, the only permitted value is 0b0001.

From Armv8.6, the only permitted value is 0b0010.

Virtualization, bits [15:12]

Virtualization support. Defined values are:

| Virtualization | Meaning |
|----------------|--|
| 0b0000 | EL2, Hyp mode, and the HVC instruction not implemented. |
| 0b0001 | EL2, Hyp mode, the HVC instruction, and all the features described by Virt_frac == 0b0001 implemented. |

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL2 is not implemented.
- 0b0001 when EL2 is implemented.

In an implementation that includes EL2, if EL2 cannot use AArch32 but EL1 can use AArch32 then this field has the value 0b0001.

If EL1 cannot use AArch32 then this field has the value 0b0000.

Note

The ID_ISARs do not identify whether the HVC instruction is implemented.

MProgMod, bits [11:8]

M-profile programmers' model support. Defined values are:

| MProgMod | Meaning |
|----------|---|
| 0b0000 | Not supported. |
| 0b0010 | Support for two-stack programmers' model. |

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

Security, bits [7:4]

Security support. Defined values are:

| Security | Meaning |
|----------|--|
| 0b0000 | EL3, Monitor mode, and the SMC instruction not implemented. |
| 0b0001 | EL3, Monitor mode, the SMC instruction, and all the features described by Sec_frac == 0b0001 implemented. |
| 0b0010 | As for 0b0001, and adds the ability to set the NSACR .RFR bit. Not permitted in Armv8 as the NSACR .RFR bit is RES0. |

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL3 is not implemented.
- 0b0001 when EL3 is implemented.

In an implementation that includes EL3, if EL3 cannot use AArch32 but EL1 can use AArch32 then this field has the value 0b0001.

If EL1 cannot use AArch32 then this field has the value 0b0000.

ProgMod, bits [3:0]

Support for the standard programmers' model for Armv4 and later. Model must support User, FIQ, IRQ, Supervisor, Abort, Undefined, and System modes. Defined values are:

| ProgMod | Meaning |
|---------|----------------|
| 0b0000 | Not supported. |
| 0b0001 | Supported. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0001 and 0b0000.

If EL1 cannot use AArch32 then this field has the value 0b0000.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_PFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_PFR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_PFR1_EL1;
elseif PSTATE.EL == EL2 then
    return ID_PFR1_EL1;
elseif PSTATE.EL == EL3 then
    return ID_PFR1_EL1;

```

ID_PFR2_EL1, AArch32 Processor Feature Register 2

The ID_PFR2_EL1 characteristics are:

Purpose

Gives information about the AArch32 programmers' model.

Must be interpreted with [ID_PFR0_EL1](#) and [ID_PFR1_EL1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register ID_PFR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ID_PFR2\[31:0\]](#).

Attributes

ID_PFR2_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|------|----|----|----|------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | RAS_frac | | | | SSBS | | | | CSV3 | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:12]

Reserved, RES0.

RAS_frac, bits [11:8]

RAS Extension fractional field. Defined values are:

| RAS_frac | Meaning |
|----------|---|
| 0b0000 | If ID_PFR0_EL1 .RAS == 0b0001, RAS Extension implemented. |
| 0b0001 | If ID_PFR0_EL1 .RAS == 0b0001, as 0b0000 and adds support for additional ERXMISC<m> System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS and support for the optional RAS Timestamp Extension. |

All other values are reserved.

This field is valid only if [ID_PFR0_EL1](#).RAS == 0b0001.

SSBS, bits [7:4]

Speculative Store Bypassing controls in AArch64 state. Defined values are:

| SSBS | Meaning |
|--------|--|
| 0b0000 | AArch32 provides no mechanism to control the use of Speculative Store Bypassing. |
| 0b0001 | AArch32 provides the PSTATE.SSBS mechanism to mark regions that are Speculative Store Bypass Safe. |

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

All other values are reserved.

CSV3, bits [3:0]

Speculative use of faulting data. Defined values are:

| CSV3 | Meaning |
|--------|--|
| 0b0000 | This PE does not disclose whether data loaded under speculation with a permission or domain fault can be used to form an address or generate condition codes or SVE predicate values to be used by other instructions in the speculative sequence. |
| 0b0001 | Data loaded under speculation with a permission or domain fault cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence. The execution timing of any other instructions in the speculative sequence is not a function of the data loaded under speculation. |

All other values are reserved.

FEAT_CSV3 implements the functionality identified by the value 0b0001.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

If FEAT_E0PD is implemented, FEAT_CSV3 must be implemented.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing ID_PFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ID_PFR2_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0011 | 0b100 |

```
if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ID_PFR2_EL1;
elseif PSTATE.EL == EL2 then
    return ID_PFR2_EL1;
elseif PSTATE.EL == EL3 then
    return ID_PFR2_EL1;
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

IFSR32_EL2, Instruction Fault Status Register (EL2)

The IFSR32_EL2 characteristics are:

Purpose

Allows access to the AArch32 [IFSR](#) register from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

AArch64 System register IFSR32_EL2 bits [31:0] are architecturally mapped to AArch32 System register [IFSR\[31:0\]](#).

This register is present only when EL1 is capable of using AArch32. Otherwise, direct accesses to IFSR32_EL2 are UNDEFINED.

If EL2 is not implemented but EL3 is implemented, and EL1 is capable of using AArch32, then this register is not RES0.

Attributes

IFSR32_EL2 is a 64-bit register.

Field descriptions

When TTBCR.EAE == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|------|----|----|----|-----|------|----|----|----|-------|------|----|----|----|------|--|--|--|---------|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | FnV | RES0 | | | | ExT | RES0 | | | | FS[4] | LPAE | | | | RES0 | | | | FS[3:0] | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | |

Bits [63:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|--|
| 0b0 | IFAR is valid. |
| 0b1 | IFAR is not valid, and holds an UNKNOWN value. |

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Prefetch Abort exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:13]

Reserved, RES0.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

FS, bits [10, 3:0]

Fault Status bits. Bits [10] and [3:0] are interpreted together.

| FS | Meaning | Applies when |
|---------|--|----------------------------------|
| 0b00001 | PC alignment fault. | |
| 0b00010 | Debug exception. | |
| 0b00011 | Access flag fault, level 1. | |
| 0b00101 | Translation fault, level 1. | |
| 0b00110 | Access flag fault, level 2. | |
| 0b00111 | Translation fault, level 2. | |
| 0b01000 | Synchronous External abort, not on translation table walk. | |
| 0b01001 | Domain fault, level 1. | |
| 0b01011 | Domain fault, level 2. | |
| 0b01100 | Synchronous External abort, on translation table walk, level 1. | |
| 0b01101 | Permission fault, level 1. | |
| 0b01110 | Synchronous External abort, on translation table walk, level 2. | |
| 0b01111 | Permission fault, level 2. | |
| 0b10000 | TLB conflict abort. | |
| 0b10100 | IMPLEMENTATION DEFINED fault (Lockdown fault). | |
| 0b11001 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b11100 | Synchronous parity or ECC error on translation table walk, level 1. | When FEAT_RAS is not implemented |
| 0b11110 | Synchronous parity or ECC error on translation table walk, level 2. | When FEAT_RAS is not implemented |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Short-descriptor translation table lookup'.

The FS field is split as follows:

- FS[4] is IFSR32_EL2[10].
- FS[3:0] is IFSR32_EL2[3:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

| LPAAE | Meaning |
|-------|---|
| 0b0 | Using the Short-descriptor translation table formats. |
| 0b1 | Using the Long-descriptor translation table formats. |

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:4]

Reserved, RES0.

When TTBCR.EAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|------|----|----|----|-----|------|----|-------|------|----|----|--------|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | FnV | RES0 | | | | ExT | RES0 | | LPAAE | RES0 | | | STATUS | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|--|
| 0b0 | IFAR is valid. |
| 0b1 | IFAR is not valid, and holds an UNKNOWN value. |

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Prefetch Abort exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:13]

Reserved, RES0.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

| LPAE | Meaning |
|-------------|---|
| 0b0 | Using the Short-descriptor translation table formats. |
| 0b1 | Using the Long-descriptor translation table formats. |

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status bits. Possible values of this field are:

| STATUS | Meaning | Applies when |
|---------------|--|----------------------------------|
| 0b000000 | Address size fault in translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010000 | Synchronous External abort, not on translation table walk. | |
| 0b010101 | Synchronous External abort on translation table walk, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk, level 3. | |
| 0b011000 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b011101 | Synchronous parity or ECC error on memory access on translation table walk, level 1. | When FEAT_RAS is not implemented |
| 0b011110 | Synchronous parity or ECC error on memory access on translation table walk, level 2. | When FEAT_RAS is not implemented |
| 0b011111 | Synchronous parity or ECC error on memory access on translation table walk, level 3. | When FEAT_RAS is not implemented |
| 0b100001 | PC alignment fault. | |
| 0b100010 | Debug exception. | |
| 0b110000 | TLB conflict abort. | |

All other values are reserved.

When FEAT_RAS is implemented, 0b011000, 0b011101, 0b011110, and 0b011111 are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Long-descriptor translation table lookup'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing IFSR32_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, IFSR32_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0000 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return IFSR32_EL2;
elsif PSTATE.EL == EL3 then
    return IFSR32_EL2;
```

MSR IFSR32_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0000 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    IFSR32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    IFSR32_EL2 = X[t];
```

ISR_EL1, Interrupt Status Register

The ISR_EL1 characteristics are:

Purpose

Shows the pending status of the IRQ, FIQ, or SError interrupt.

When executing at EL2, EL3 or Secure EL1 when [SCR_EL3.EEL2](#) == 0b0, this shows the pending status of the physical IRQ, FIQ, or SError interrupts.

When executing at either Non-secure EL1 or at Secure EL1 when [SCR_EL3.EEL2](#) == 0b1:

- If the [HCR_EL2](#).{IMO,FMO,AMO} bit has a value of 1, the corresponding ISR_EL1.{I,F,A} bit shows the pending status of the virtual IRQ, FIQ, or SError.
- If the [HCR_EL2](#).{IMO,FMO,AMO} bit has a value of 0, the corresponding ISR_EL1.{I,F,A} bit shows the pending status of the physical IRQ, FIQ, or SError.

Configuration

AArch64 System register ISR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [ISR\[31:0\]](#).

Attributes

ISR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|--|--|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | IS | | FS | | A | | I | | F | | RES0 | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | |

Bits [63:11]

Reserved, RES0.

IS, bit [10]

When FEAT_NMI is implemented:

IRQ with Superpriority pending bit. Indicates whether an IRQ interrupt with Superpriority is pending.

| IS | Meaning |
|-----|---|
| 0b0 | No pending IRQ with Superpriority. |
| 0b1 | An IRQ interrupt with Superpriority is pending. |

Note

The function of ISR_EL1.I to indicate the presence of a pending IRQ interrupt is unchanged regardless of Superpriority.

Otherwise:

Reserved, RES0.

FS, bit [9]**When FEAT_NMI is implemented:**

FIQ with Superpriority pending bit. Indicates whether an FIQ interrupt with Superpriority is pending.

| FS | Meaning |
|-----------|---|
| 0b0 | No pending FIQ with Superpriority. |
| 0b1 | An FIQ interrupt with Superpriority is pending. |

Note

The function of ISR_EL1.F to indicate the presence of a pending FIQ interrupt is unchanged regardless of Superpriority.

Otherwise:

Reserved, RES0.

A, bit [8]

SError interrupt pending bit. Indicates whether an SError interrupt is pending.

| A | Meaning |
|----------|---------------------------------|
| 0b0 | No pending SError. |
| 0b1 | An SError interrupt is pending. |

If the SError interrupt is edge-triggered, this field is cleared to zero when the physical SError interrupt is taken.

I, bit [7]

IRQ pending bit. Indicates whether an IRQ interrupt is pending.

| I | Meaning |
|----------|------------------------------|
| 0b0 | No pending IRQ. |
| 0b1 | An IRQ interrupt is pending. |

F, bit [6]

FIQ pending bit. Indicates whether an FIQ interrupt is pending.

| F | Meaning |
|----------|------------------------------|
| 0b0 | No pending FIQ. |
| 0b1 | An FIQ interrupt is pending. |

Bits [5:0]

Reserved, RES0.

Accessing ISR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ISR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b000 | 0b1100 | 0b0001 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.ISR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return ISR_EL1;
elsif PSTATE.EL == EL2 then
    return ISR_EL1;
elsif PSTATE.EL == EL3 then
    return ISR_EL1;
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

LORC_EL1, LORegion Control (EL1)

The LORC_EL1 characteristics are:

Purpose

Enables and disables LORegions, and selects the current LORegion descriptor.

Configuration

This register is present only when FEAT_LOR is implemented. Otherwise, direct accesses to LORC_EL1 are UNDEFINED.

If no LORegion descriptors are supported by the PE, then this register is RES0.

Attributes

LORC_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | DS | | | | | | | | | | RES0 | | EN | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:10]

Reserved, RES0.

DS, bits [9:2]

Descriptor Select. Selects the current LORegion descriptor accessed by [LORSA_EL1](#), [LOREA_EL1](#), and [LORN_EL1](#).

The number of LORegion descriptors in IMPLEMENTATION DEFINED. The maximum number of LORegion descriptors supported is 256. If the number is less than 256, then bits[63:M+2] are RES0, where M is $\text{Log}_2(\text{Number of LORegion descriptors supported by the implementation})$.

If this field points to an LORegion descriptor that is not supported by an implementation, then the registers [LORN_EL1](#), [LOREA_EL1](#), and [LORSA_EL1](#) are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [1]

Reserved, RES0.

EN, bit [0]

Enable. Indicates whether LORegions are enabled.

| EN | Meaning |
|-----|---|
| 0b0 | Disabled. Memory accesses do not match any LORegions. |
| 0b1 | Enabled. Memory accesses may match a LORegion. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing LORC_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, LORC_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.LORC_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return LORC_EL1;
    elsif PSTATE.EL == EL2 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return LORC_EL1;
    elsif PSTATE.EL == EL3 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        else
            return LORC_EL1;

```

MSR LORC_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.LORC_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            LORC_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                LORC_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        else
            LORC_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

LOREA_EL1, LORegion End Address (EL1)

The LOREA_EL1 characteristics are:

Purpose

Holds the physical address of the end of the LORegion described in the current LORegion descriptor selected by [LORC_EL1](#).DS.

Configuration

This register is present only when FEAT_LOR is implemented. Otherwise, direct accesses to LOREA_EL1 are UNDEFINED.

This register is RES0 if any of the following apply:

- No LORegion descriptors are supported by the PE.
- [LORC_EL1](#).DS points to a LORegion that is not supported by the PE.

Attributes

LOREA_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | EA[51:48] | | | | EA[47:16] | | | | | | | | | | | | | | | |
| EA[47:16] | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Any of the fields in this register are permitted to be cached in a TLB.

Bits [63:52]

Reserved, RES0.

EA[51:48], bits [51:48]

When FEAT_LPA is implemented:

Extension to EA[47:16]. For more information, see EA[47:16].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EA[47:16], bits [47:16]

Bits [47:16] of the end physical address of an LORegion described in the current LORegion descriptor selected by [LORC_EL1](#).DS. Bits[15:0] of this address are 0xFFFF. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When FEAT_LPA is implemented and 52-bit addresses are in use, EA[51:48] form bits [51:48] of the end physical address of the LORegion. Otherwise, when 52-bit addresses are not in use, EA[51:48] is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:0]

Reserved, RES0.

Accessing LOREA_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, LOREA_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.LOREA_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return LOREA_EL1;
    elsif PSTATE.EL == EL2 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return LOREA_EL1;
    elsif PSTATE.EL == EL3 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        else
            return LOREA_EL1;

```

MSR LOREA_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.LOREA_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            LOREA_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            LOREA_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        LOREA_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

LORID_EL1, LORegionID (EL1)

The LORID_EL1 characteristics are:

Purpose

Indicates the number of LORegions and LORegion descriptors supported by the PE.

Configuration

This register is present only when FEAT_LOR is implemented. Otherwise, direct accesses to LORID_EL1 are UNDEFINED.

If no LORegion descriptors are implemented, then the registers [LORC_EL1](#), [LORN_EL1](#), [LOREA_EL1](#), and [LORSA_EL1](#) are RES0.

Attributes

LORID_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | LD | | | | | | | | RES0 | | | | | | | | LR | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

LD, bits [23:16]

Number of LORegion descriptors supported by the PE. This is an 8-bit binary number.

Bits [15:8]

Reserved, RES0.

LR, bits [7:0]

Number of LORegions supported by the PE. This is an 8-bit binary number.

Note

If LORID_EL1 indicates that no LORegions are implemented, then LoadLOAcquire and StoreLORelease will behave as LoadAcquire and StoreRelease.

Accessing LORID_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, LORID_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.LORID_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return LORID_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return LORID_EL1;
    elsif PSTATE.EL == EL3 then
        return LORID_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

LORN_EL1, LORegion Number (EL1)

The LORN_EL1 characteristics are:

Purpose

Holds the number of the LORegion described in the current LORegion descriptor selected by [LORC_EL1](#).DS.

Configuration

This register is present only when FEAT_LOR is implemented. Otherwise, direct accesses to LORN_EL1 are UNDEFINED.

This register is RES0 if any of the following apply:

- No LORegion descriptors are supported by the PE.
- [LORC_EL1](#).DS points to a LORegion that is not supported by the PE.

Attributes

LORN_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | Num | | | | | | | | | | | | | | | | | | | | | | | |

Any of the fields in this register are permitted to be cached in a TLB.

Bits [63:8]

Reserved, RES0.

Num, bits [7:0]

Number of the LORegion described in the current LORegion descriptor selected by [LORC_EL1](#).DS.

The maximum number of LORegions supported by the PE is 256. If the maximum number is less than 256, then bits[8:N] are RES0, where N is (Log₂(Number of LORegions supported by the PE)).

If this field points to a LORegion that is not supported by the PE, then the current LORegion descriptor does not match any LORegion.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing LORN_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, LORN_EL1

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b010 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.LORN_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return LORN_EL1;
    elsif PSTATE.EL == EL2 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return LORN_EL1;
    elsif PSTATE.EL == EL3 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        else
            return LORN_EL1;

```

MSR LORN_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.LORN_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            LORN_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            LORN_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        else
            LORN_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

LORSA_EL1, LORegion Start Address (EL1)

The LORSA_EL1 characteristics are:

Purpose

Indicates whether the current LORegion descriptor selected by [LORC_EL1](#).DS is enabled, and holds the physical address of the start of the LORegion.

Configuration

This register is present only when FEAT_LOR is implemented. Otherwise, direct accesses to LORSA_EL1 are UNDEFINED.

This register is RES0 if any of the following apply:

- No LORegion descriptors are supported by the PE.
- [LORC_EL1](#).DS points to a LORegion that is not supported by the PE.

Attributes

LORSA_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | SA | | | | | | | | | | | | | | | | | | | | |
| SA | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | Valid |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | | | | | | | | | | | | | | | | Valid | | | | | | | | | | | | | | | | |

Any of the fields in this register are permitted to be cached in a TLB.

Bits [63:52]

Reserved, RES0.

SA, bits [51:16]

SA encoding when FEAT_LPA is implemented

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

SA, bits [35:0]

Bits [51:16] of the start physical address of the LORegion described in the current LORegion descriptor selected by [LORC_EL1](#).DS.

Bits[15:0] of this address are 0x0000.

For implementations with fewer than 52 physical address bits, the corresponding upper bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SA encoding when FEAT_LPA is not implemented

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | SA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [35:32]

Reserved, RES0.

SA, bits [31:0]

Bits [47:16] of the start physical address of the LORegion described in the current LORegion descriptor selected by [LORC_EL1](#).DS.

Bits[15:0] of this address are 0x0000.

For implementations with fewer than 48 physical address bits, the corresponding upper bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:1]

Reserved, RES0.

Valid, bit [0]

Indicates whether the current LORegion descriptor is enabled.

| Valid | Meaning |
|-------|----------------------------------|
| 0b0 | LORegion descriptor is disabled. |
| 0b1 | LORegion descriptor is enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing LORSA_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, LORSA_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.LORSA_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return LORSA_EL1;
    elsif PSTATE.EL == EL2 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return LORSA_EL1;
    elsif PSTATE.EL == EL3 then
        if SCR_EL3.NS == '0' then
            UNDEFINED;
        else
            return LORSA_EL1;

```

MSR LORSA_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.TLOR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.LORSA_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            LORSA_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.TLOR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.TLOR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            LORSA_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        UNDEFINED;
    else
        LORSA_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MAIR_EL1, Memory Attribute Indirection Register (EL1)

The MAIR_EL1 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIdx values in a Long-descriptor format translation table entry for stage 1 translations at EL1.

Configuration

AArch64 System register MAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [PRRR\[31:0\]](#) when TTBCR.EAE == 0.

AArch64 System register MAIR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [MAIR0\[31:0\]](#) when TTBCR.EAE == 1.

AArch64 System register MAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register [NMRR\[31:0\]](#) when TTBCR.EAE == 0.

AArch64 System register MAIR_EL1 bits [63:32] are architecturally mapped to AArch32 System register [MAIR1\[31:0\]](#) when TTBCR.EAE == 1.

Attributes

MAIR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------|----|----|----|----|----|----|----|-----------------------|----|----|----|----|----|----|----|-----------------------|----|----|----|----|----|----|----|-----------------------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Attr7 | | | | | | | | Attr6 | | | | | | | | Attr5 | | | | | | | | Attr4 | | | | | | | |
| Attr3 | | | | | | | | Attr2 | | | | | | | | Attr1 | | | | | | | | Attr0 | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

MAIR_EL1 is permitted to be cached in a TLB.

Attr<n>, bits [8n+7:8n], for n = 7 to 0

The memory attribute encoding for an AttrIdx[2:0] entry in a Long descriptor format translation table entry, where AttrIdx[2:0] gives the value of <n> in Attr<n>.

Attr is encoded as follows:

| Attr | Meaning |
|--|--|
| 0b0000dd00 | Device memory. See encoding of 'dd' for the type of Device memory. |
| 0b0000dd01 | If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of 'dd' for the type of Device memory. Otherwise, UNPREDICTABLE. |
| 0b0000dd1x | UNPREDICTABLE. |
| 0boooooiii, (oooo != 0000 and iiii != 0000) | Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal Memory. |
| 0b01000000 | If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE. |
| 0b10100000 | If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE. |
| 0b11110000 | If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE. |
| 0bxxxx0000, (xxxx != 0000, xxxx != 0100, xxxx != 1010, xxxx != 1111) | UNPREDICTABLE. |

'dd' is encoded as follows:

| dd | Meaning |
|------|----------------------|
| 0b00 | Device-nGnRnE memory |
| 0b01 | Device-nGnRE memory |
| 0b10 | Device-nGRE memory |
| 0b11 | Device-GRE memory |

'oooo' is encoded as follows:

| 'oooo' | Meaning |
|------------------------|--|
| 0b0000 | See encoding of Attr |
| 0b00RW, RW not 0b00 | Normal memory, Outer Write-Through Transient |
| 0b0100 | Normal memory, Outer Non-cacheable |
| 0b01RW, RW not 0b00 | Normal memory, Outer Write-Back Transient |
| 0b10RW | Normal memory, Outer Write-Through Non-transient |
| 0b11RW | Normal memory, Outer Write-Back Non-transient |

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

'iiii' is encoded as follows:

| 'iiii' | Meaning |
|------------------------|--|
| 0b0000 | See encoding of Attr |
| 0b00RW, RW not 0b00 | Normal memory, Inner Write-Through Transient |
| 0b0100 | Normal memory, Inner Non-cacheable |
| 0b01RW, RW not 0b00 | Normal memory, Inner Write-Back Transient |
| 0b10RW | Normal memory, Inner Write-Through Non-transient |
| 0b11RW | Normal memory, Inner Write-Back Non-transient |

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in 'oooo' and 'iiii' fields have the following meanings:

| R or W | Meaning |
|--------|-------------|
| 0b0 | No Allocate |
| 0b1 | Allocate |

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic MAIR_EL1 or MAIR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MAIR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.MAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x140];
    else
        return MAIR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return MAIR_EL2;
    else
        return MAIR_EL1;
elsif PSTATE.EL == EL3 then
    return MAIR_EL1;

```

MSR MAIR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.MAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x140] = X[t];
    else
        MAIR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' then
            MAIR_EL2 = X[t];
        else
            MAIR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        MAIR_EL1 = X[t];

```

MRS <Xt>, MAIR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x140];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' then
            return MAIR_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
            return MAIR_EL1;
        else
            UNDEFINED;

```

MSR MAIR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x140] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        MAIR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        MAIR_EL1 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MAIR_EL2, Memory Attribute Indirection Register (EL2)

The MAIR_EL2 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIdx values in a Long-descriptor format translation table entry for stage 1 translations at EL2.

Configuration

AArch64 System register MAIR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HMAIRO\[31:0\]](#).

AArch64 System register MAIR_EL2 bits [63:32] are architecturally mapped to AArch32 System register [HMAIR1\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MAIR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Attr7 | | | | | | | | Attr6 | | | | | | | | Attr5 | | | | | | | | Attr4 | | | | | | | |
| Attr3 | | | | | | | | Attr2 | | | | | | | | Attr1 | | | | | | | | Attr0 | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

MAIR_EL2 is permitted to be cached in a TLB.

Attr<n>, bits [8n+7:8n], for n = 7 to 0

The memory attribute encoding for an AttrIdx[2:0] entry in a Long descriptor format translation table entry, where AttrIdx[2:0] gives the value of <n> in Attr<n>.

Attr is encoded as follows:

| Attr | Meaning |
|--|--|
| 0b0000dd00 | Device memory. See encoding of 'dd' for the type of Device memory. |
| 0b0000dd01 | If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of 'dd' for the type of Device memory. Otherwise, UNPREDICTABLE. |
| 0b0000dd1x | UNPREDICTABLE. |
| 0boooooiii, (oooo != 0000 and iiii != 0000) | Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal Memory. |
| 0b01000000 | If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE. |
| 0b10100000 | If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE. |
| 0b11110000 | If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE. |
| 0bxxxx0000, (xxxx != 0000, xxxx != 0100, xxxx != 1010, xxxx != 1111) | UNPREDICTABLE. |

'dd' is encoded as follows:

| dd | Meaning |
|------|----------------------|
| 0b00 | Device-nGnRnE memory |
| 0b01 | Device-nGnRE memory |
| 0b10 | Device-nGRE memory |
| 0b11 | Device-GRE memory |

'oooo' is encoded as follows:

| 'oooo' | Meaning |
|------------------------|--|
| 0b0000 | See encoding of Attr |
| 0b00RW, RW not 0b00 | Normal memory, Outer Write-Through Transient |
| 0b0100 | Normal memory, Outer Non-cacheable |
| 0b01RW, RW not 0b00 | Normal memory, Outer Write-Back Transient |
| 0b10RW | Normal memory, Outer Write-Through Non-transient |
| 0b11RW | Normal memory, Outer Write-Back Non-transient |

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

'iiii' is encoded as follows:

| 'iiii' | Meaning |
|------------------------|--|
| 0b0000 | See encoding of Attr |
| 0b00RW, RW not 0b00 | Normal memory, Inner Write-Through Transient |
| 0b0100 | Normal memory, Inner Non-cacheable |
| 0b01RW, RW not 0b00 | Normal memory, Inner Write-Back Transient |
| 0b10RW | Normal memory, Inner Write-Through Non-transient |
| 0b11RW | Normal memory, Inner Write-Back Non-transient |

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in 'oooo' and 'iiii' fields have the following meanings:

| R or W | Meaning |
|---------------|----------------|
| 0b0 | No Allocate |
| 0b1 | Allocate |

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic MAIR_EL2 or MAIR_EL1 is not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MAIR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b100 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return MAIR_EL2;
elsif PSTATE.EL == EL3 then
    return MAIR_EL2;

```

MSR MAIR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b100 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    MAIR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MAIR_EL2 = X[t];

```

MRS <Xt>, MAIR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b000 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.MAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x140];
    else
        return MAIR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return MAIR_EL2;
    else
        return MAIR_EL1;
elsif PSTATE.EL == EL3 then
    return MAIR_EL1;

```

MSR MAIR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.MAIR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x140] = X[t];
    else
        MAIR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        MAIR_EL2 = X[t];
    else
        MAIR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    MAIR_EL1 = X[t];

```

MAIR_EL3, Memory Attribute Indirection Register (EL3)

The MAIR_EL3 characteristics are:

Purpose

Provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations at EL3.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to MAIR_EL3 are UNDEFINED.

Attributes

MAIR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Attr7 | | | | | | | | Attr6 | | | | | | | | Attr5 | | | | | | | | Attr4 | | | | | | | |
| Attr3 | | | | | | | | Attr2 | | | | | | | | Attr1 | | | | | | | | Attr0 | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

MAIR_EL3 is permitted to be cached in a TLB.

Attr<n>, bits [8n+7:8n], for n = 7 to 0

The memory attribute encoding for an AttrIndx[2:0] entry in a Long descriptor format translation table entry, where AttrIndx[2:0] gives the value of <n> in Attr<n>.

Attr is encoded as follows:

| Attr | Meaning |
|--|--|
| 0b0000dd00 | Device memory. See encoding of 'dd' for the type of Device memory. |
| 0b0000dd01 | If FEAT_XS is implemented: Device memory with the XS attribute set to 0. See encoding of 'dd' for the type of Device memory. Otherwise, UNPREDICTABLE. |
| 0b0000dd1x | UNPREDICTABLE. |
| 0boooooiii, (oooo != 0000 and iiii != 0000) | Normal memory. See encoding of 'oooo' and 'iiii' for the type of Normal Memory. |
| 0b01000000 | If FEAT_XS is implemented: Normal Inner Non-cacheable, Outer Non-cacheable memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE. |
| 0b10100000 | If FEAT_XS is implemented: Normal Inner Write-through Cacheable, Outer Write-through Cacheable, Read-Allocate, No-Write Allocate, Non-transient memory with the XS attribute set to 0. Otherwise, UNPREDICTABLE. |
| 0b11110000 | If FEAT_MTE2 is implemented: Tagged Normal Inner Write-Back, Outer Write-Back, Read-Allocate, Write-Allocate Non-transient memory. Otherwise, UNPREDICTABLE. |
| 0bxxxx0000, (xxxx != 0000, xxxx != 0100, xxxx != 1010, xxxx != 1111) | UNPREDICTABLE. |

'dd' is encoded as follows:

| dd | Meaning |
|------|----------------------|
| 0b00 | Device-nGnRnE memory |
| 0b01 | Device-nGnRE memory |
| 0b10 | Device-nGRE memory |
| 0b11 | Device-GRE memory |

'oooo' is encoded as follows:

| 'oooo' | Meaning |
|------------------------|--|
| 0b0000 | See encoding of Attr |
| 0b00RW, RW not 0b00 | Normal memory, Outer Write-Through Transient |
| 0b0100 | Normal memory, Outer Non-cacheable |
| 0b01RW, RW not 0b00 | Normal memory, Outer Write-Back Transient |
| 0b10RW | Normal memory, Outer Write-Through Non-transient |
| 0b11RW | Normal memory, Outer Write-Back Non-transient |

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

'iiii' is encoded as follows:

| 'iiii' | Meaning |
|------------------------|--|
| 0b0000 | See encoding of Attr |
| 0b00RW, RW not 0b00 | Normal memory, Inner Write-Through Transient |
| 0b0100 | Normal memory, Inner Non-cacheable |
| 0b01RW, RW not 0b00 | Normal memory, Inner Write-Back Transient |
| 0b10RW | Normal memory, Inner Write-Through Non-transient |
| 0b11RW | Normal memory, Inner Write-Back Non-transient |

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in 'oooo' and 'iiii' fields have the following meanings:

| R or W | Meaning |
|---------------|----------------|
| 0b0 | No Allocate |
| 0b1 | Allocate |

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MAIR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b110 | 0b1010 | 0b0010 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return MAIR_EL3;
```

MSR MAIR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b110 | 0b1010 | 0b0010 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    MAIR_EL3 = X[t];
```

MDCCINT_EL1, Monitor DCC Interrupt Enable Register

The MDCCINT_EL1 characteristics are:

Purpose

Enables interrupt requests to be signaled based on the DCC status flags.

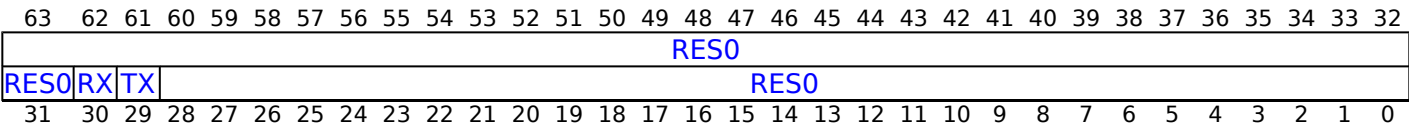
Configuration

AArch64 System register MDCCINT_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGDCCINTI\[31:0\]](#).

Attributes

MDCCINT_EL1 is a 64-bit register.

Field descriptions



Bits [63:31]

Reserved, RES0.

RX, bit [30]

DCC interrupt request enable control for DTRRX. Enables a common **COMMIRQ** interrupt request to be signaled based on the DCC status flags.

| RX | Meaning |
|-----|---|
| 0b0 | No interrupt request generated by DTRRX. |
| 0b1 | Interrupt request will be generated on RXfull == 1. |

If legacy **COMMRX** and **COMMTX** signals are implemented, then these are not affected by the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

TX, bit [29]

DCC interrupt request enable control for DTRTX. Enables a common **COMMIRQ** interrupt request to be signaled based on the DCC status flags.

| TX | Meaning |
|-----|---|
| 0b0 | No interrupt request generated by DTRTX. |
| 0b1 | Interrupt request will be generated on TXfull == 0. |

If legacy **COMMRX** and **COMMTX** signals are implemented, then these are not affected by the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [28:0]

Reserved, RES0.

Accessing MDCCINT_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MDCCINT_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    return MDCCINT_EL1;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MDCCINT_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MDCCINT_EL1;
elsif PSTATE.EL == EL3 then
    return MDCCINT_EL1;

```

MSR MDCCINT_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    MDCCINT_EL1 = X[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCCINT_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCCINT_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    MDCCINT_EL1 = X[t];

```

MDCCSR_EL0, Monitor DCC Status Register

The MDCCSR_EL0 characteristics are:

Purpose

Read-only register containing control status flags for the DCC.

Configuration

AArch64 System register MDCCSR_EL0 bits [30:29] are architecturally mapped to External register [EDSCR\[30:29\]](#).

AArch64 System register MDCCSR_EL0 bits [30:29] are architecturally mapped to AArch32 System register [DBGDSCRint\[30:29\]](#).

Attributes

MDCCSR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|--------|--------|------|----|----|----|----|----|----|----|----|----|----|-----|----|----|------|-----|------|----|----|----|----|----|----|-----|----|----|------|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | RXfull | TXfull | RES0 | | | | | | | | | | | RAZ | | | RES0 | RAZ | RES0 | | | | | | | RAZ | | | RES0 | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:31]

Reserved, RES0.

RXfull, bit [30]

DTRRX full. Read-only view of the equivalent bit in the [EDSCR](#).

TXfull, bit [29]

DTRTX full. Read-only view of the equivalent bit in the [EDSCR](#).

Bits [28:19]

Reserved, RES0.

Bits [18:15]

Reserved, RAZ.

Bits [14:13]

Reserved, RES0.

Bit [12]

Reserved, RAZ.

Bits [11:6]

Reserved, RES0.

Bits [5:2]

Reserved, RAZ.

Bits [1:0]

Reserved, RES0.

Accessing MDCCSR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MDCCSR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b011 | 0b0000 | 0b0001 | 0b000 |

```

if Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    return MDCCSR_EL0;
elsif PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif MDCR_EL1.TDCC == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> != '00') then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MDCCSR_EL0;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MDCCSR_EL0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MDCCSR_EL0;

```

```
else
    return MDCCSR_EL0;
elsif PSTATE.EL == EL3 then
    return MDCCSR_EL0;
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MDCR_EL2, Monitor Debug Configuration Register (EL2)

The MDCR_EL2 characteristics are:

Purpose

Provides EL2 configuration options for self-hosted debug and the Performance Monitors Extension.

Configuration

AArch64 System register MDCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HDCR\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MDCR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|--------|-------|------|------|------|------|------|------|------|------|------|------|------|------|-------|-------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | HPMFZO | MTPME | TDCC | HLPE | E2TB | HCCD | RES0 | TTRF | RES0 | HPMD | RES0 | TPMS | E2PB | TDRA | TDOSA | TDATD | HPME | TPM | TPM | TPM | TPM | TPM | TPM | TPM | TPM | TPM | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | | |

Bits [63:37]

Reserved, RES0.

HPMFZS, bit [36]

When FEAT_SPEv1p2 is implemented:

Hyp Performance Monitors Freeze-on-SPE event. Stop counters when [PMBLIMITR_EL1](#).{PMFZ, E} == {1, 1} and [PMBSR_EL1](#).S == 1.

| HPMFZS | Meaning |
|--------|--|
| 0b0 | Do not freeze on Statistical Profiling Buffer Management event. |
| 0b1 | Event counters do not count following a Statistical Profiling Buffer Management event. |

If MDCR_EL2.HPMN is less than [PMCR_EL0.N](#), this field affects the operation of event counters in the range [MDCR_EL2.HPMN .. ([PMCR_EL0.N](#)-1)].

This field does not affect the operation of other event counters and [PMCCNTR_ELO](#).

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [35:30]

Reserved, RES0.

HPMFZO, bit [29]

When FEAT_PMUv3p7 is implemented:

Hyp Performance Monitors Freeze-on-overflow. Stop event counters on overflow.

| HPMFZO | Meaning |
|--------|---|
| 0b0 | Do not freeze on overflow. |
| 0b1 | Event counters do not count when PMOVSLR_EL0 [(PMCR_EL0 .N-1):MDCR_EL2.HPMN] is nonzero. |

If MDCR_EL2.HPMN is less than [PMCR_EL0](#).N, this field affects the operation of event counters in the range [MDCR_EL2.HPMN .. ([PMCR_EL0](#).N-1)].

This field does not affect the operation of other event counters and [PMCCNTR_EL0](#).

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTPME, bit [28]

When FEAT_MTPMU is implemented and EL3 is not implemented:

Multi-threaded PMU Enable. Enables use of the [PMEVTYPER<n>_EL0](#).MT bits.

| MTPME | Meaning |
|-------|--|
| 0b0 | FEAT_MTPMU is disabled. The Effective value of PMEVTYPER<n>_EL0 .MT is zero. |
| 0b1 | PMEVTYPER<n>_EL0 .MT bits not affected by this field. |

If FEAT_MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is IMPLEMENTATION DEFINED whether the PE behaves as if this field is 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to 1.

Otherwise:

Reserved, RES0.

TDCC, bit [27]

When FEAT_FGT is implemented:

Trap DCC. Traps use of the Debug Comms Channel at EL1 and EL0 to EL2.

| TDCC | Meaning |
|-------------|--|
| 0b0 | This control does not cause any register accesses to be trapped. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, accesses to the DCC registers at EL1 and EL0 generate a Trap exception to EL2, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state. |

The DCC registers trapped by this control are:

AArch64: [OSDTRRX_EL1](#), [OSDTRTX_EL1](#), [MDCCSR_EL0](#), [MDCCINT_EL1](#), and, when the PE is in Non-debug state, [DBGDTR_EL0](#), [DBGDTRRX_EL0](#), and [DBGDTRTX_EL0](#).

AArch32: [DBGDTRRXext](#), [DBGDTRTXext](#), [DBGDSCRint](#), [DBGDCCINT](#), and, when the PE is in Non-debug state, [DBGDTRRXint](#) and [DBGDTRTXint](#).

The traps are reported with EC syndrome value:

- 0x05 for trapped AArch32 MRC and MCR accesses with coproc == 0b1110.
- 0x06 for trapped AArch32 LDC to [DBGDTRTXint](#) and STC from [DBGDTRRXint](#).
- 0x18 for trapped AArch64 MRS and MSR accesses.

When the PE is in Debug state, MDCR_EL2.TDCC does not trap any accesses to:

AArch64: [DBGDTR_EL0](#), [DBGDTRRX_EL0](#), and [DBGDTRTX_EL0](#).

AArch32: [DBGDTRRXint](#) and [DBGDTRTXint](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HLP, bit [26]

When FEAT_PMUv3p5 is implemented:

Hypervisor Long event counter enable. Determines when unsigned overflow is recorded by an event counter overflow bit.

| HLP | Meaning |
|------------|--|
| 0b0 | Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0 [31:0]. |
| 0b1 | Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0 [63:0]. |

If MDCR_EL2.HPMN is less than [PMCR_EL0](#).N, this bit affects the operation of event counters in the range [MDCR_EL2.HPMN..([PMCR_EL0](#).N-1)].

This field does not affect the operation of other event counters.

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E2TB, bits [25:24]**When FEAT_TRBE is implemented:**

EL2 Trace Buffer.

If EL2 is implemented and enabled in the Trace Buffer owning Security state, controls the owning translation regime.

If EL2 is implemented and enabled in the current Security state, controls access to Trace Buffer control registers from EL1.

| E2TB | Meaning |
|-------------|--|
| 0b00 | <p>If EL2 is implemented and enabled in the Trace Buffer owning Security state, the Trace Buffer owning Exception level is EL2. Otherwise, the Trace Buffer owning Exception level is EL1 and, if <code>TraceBufferEnabled() == TRUE</code>, tracing is prohibited at EL2.</p> <p>If EL2 is implemented and enabled in the current Security state, accesses to Trace Buffer control registers at EL1 generate a Trap exception to EL2.</p> |
| 0b10 | <p>Trace Buffer owning Exception level is EL1. If <code>TraceBufferEnabled() == TRUE</code>, tracing is prohibited at EL2.</p> <p>If EL2 is implemented and enabled in the current Security state, accesses to Trace Buffer control registers at EL1 generate a Trap exception to EL2.</p> |
| 0b11 | <p>Trace Buffer owning Exception level is EL1. If <code>TraceBufferEnabled() == TRUE</code>, tracing is prohibited at EL2.</p> |

All other values are reserved.

The Trace Buffer control registers trapped by this control are: [TRBBASER_EL1](#), [TRBLIMITR_EL1](#), [TRBMAR_EL1](#), [TRBPTR_EL1](#), [TRBSR_EL1](#), and [TRBTRG_EL1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HCCD, bit [23]**When FEAT_PMUv3p5 is implemented:**

Hypervisor Cycle Counter Disable. Prohibits [PMCCNTR_EL0](#) from counting at EL2.

| HCCD | Meaning |
|-------------|--|
| 0b0 | Cycle counting by PMCCNTR_EL0 is not affected by this mechanism. |
| 0b1 | Cycle counting by PMCCNTR_EL0 is prohibited at EL2. |

This field does not affect the CPU_CYCLES event or any other event that counts cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [22:20]

Reserved, RES0.

TTRF, bit [19]

When FEAT_TRF is implemented:

Traps use of the Trace Filter Control registers at EL1 to EL2, as follows:

- Access to [TRFCR_EL1](#) is trapped to EL2, reported using EC syndrome value 0x18.
- Access to [TRFCR](#) is trapped to EL2, reported using EC syndrome value 0x03.

| TTRF | Meaning |
|------|--|
| 0b0 | Accesses to TRFCR_EL1 and TRFCR at EL1 are not affected by this control. |
| 0b1 | Accesses to TRFCR_EL1 and TRFCR at EL1 generate a trap exception to EL2 when EL2 is enabled in the current Security state. |

Otherwise:

Reserved, RES0.

Bit [18]

Reserved, RES0.

HPMD, bit [17]

When FEAT_PMUv3p1 is implemented and FEAT_Debugv8p2 is implemented:

Guest Performance Monitors Disable. Controls event counting by some event counters at EL2.

| HPMD | Meaning |
|------|--|
| 0b0 | Event counting and PMCCNTR_EL0 are not affected by this mechanism. |
| 0b1 | Event counting by some event counters is prohibited at EL2. If PMCR_EL0.DP is 1, PMCCNTR_EL0 is disabled at EL2. Otherwise, PMCCNTR_EL0 is not affected by this mechanism. |

If MDCR_EL2.HPMN is not 0, this field affects the operation of event counters in the range [0 .. (MDCR_EL2.HPMN-1)].

This field does not affect the operation of other event counters.

If [PMCR_EL0.DP](#) is 1, this field affects [PMCCNTR_EL0](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

When FEAT_PMUv3p1 is implemented:

Guest Performance Monitors Disable. Controls event counting by some event counters at EL2.

| HPMD | Meaning |
|------|---|
| 0b0 | Event counting and PMCCNTR_EL0 are not affected by this mechanism. |
| 0b1 | If ExternalSecureNoninvasiveDebugEnabled() is FALSE, event counting by some event counters is prohibited at EL2, and if PMCR_EL0.DP is 1, PMCCNTR_EL0 is disabled at EL2. |

If `ExternalSecureNoninvasiveDebugEnabled()` is TRUE, this field does not affect the event counters and does not affect [PMCCNTR_EL0](#).

Otherwise:

- If MDCR_EL2.HPMN is not 0, this field affects the operation of event counters in the range [0 .. (MDCR_EL2.HPMN-1)].
- This field does not affect the operation of other event counters.
- If [PMCR_EL0.DP](#) is 1, this field affects [PMCCNTR_EL0](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [16:15]

Reserved, RES0.

TPMS, bit [14]

When FEAT_SPE is implemented:

Trap Performance Monitor Sampling. If EL2 is implemented and enabled in the current Security state, controls access to Statistical Profiling control registers from EL1.

| TPMS | Meaning |
|------|---|
| 0b0 | Do not trap Statistical Profiling controls to EL2. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, accesses to Statistical Profiling control registers at EL1 generate a Trap exception to EL2. |

The Statistical Profiling control registers trapped by this control are:

- [PMSCR_EL1](#), [PMSEVFR_EL1](#), [PMSFCR_EL1](#), [PMSICR_EL1](#), [PMSIDR_EL1](#), [PMSIRR_EL1](#), and [PMSLATR_EL1](#).
- If FEAT_SPEv1p2 is implemented, [PMSNEVER_EL1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E2PB, bits [13:12]

When FEAT_SPE is implemented:

EL2 Profiling Buffer. If EL2 is implemented and enabled in the Profiling Buffer owning Security state, this field controls the owning translation regime. If EL2 is implemented and enabled in the current Security state, this field controls access to Profiling Buffer control registers from EL1.

| E2PB | Meaning |
|-------------|---|
| 0b00 | If EL2 is implemented and enabled in the Profiling Buffer owning Security state, the Profiling Buffer uses the EL2 or EL2&0 stage 1 translation regime. Otherwise the Profiling Buffer uses the EL1&0 stage 1 translation regime. If EL2 is implemented and enabled in the current Security state, accesses to Profiling Buffer control registers at EL1 generate a Trap exception to EL2. |
| 0b10 | Profiling Buffer uses the EL1&0 stage 1 translation regime. If EL2 is implemented and enabled in the current Security state, accesses to Profiling Buffer control registers at EL1 generate a Trap exception to EL2. |
| 0b11 | Profiling Buffer uses the EL1&0 stage 1 translation regime. Accesses to Profiling Buffer control registers at EL1 are not trapped to EL2. |

All other values are reserved.

The Profiling Buffer control registers trapped by this control are: [PMBLIMITR_EL1](#), [PMBPTR_EL1](#), and [PMBSR_EL1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TDRA, bit [11]

Trap Debug ROM Address register access. Traps System register accesses to the Debug ROM registers to EL2 when EL2 is enabled in the current Security state as follows:

- If EL1 is using AArch64 state, accesses to [MDRAR_EL1](#) are trapped to EL2, reported using EC syndrome value 0x18.
- If EL0 or EL1 is using AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x05 and MRRC or MCRR accesses are trapped to EL2, reported using EC syndrome value 0x0C:
 - [DBGDRAR](#), [DBGDSAR](#).

| TDRA | Meaning |
|-------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL0 and EL1 System register accesses to the Debug ROM registers are trapped to EL2 when EL2 is enabled in the current Security state, unless it is trapped by DBGDSCRext .UDCCdis or MDSCR_EL1 .TDCC. |

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- [MDCR_EL2](#).TDE == 1.
- [HCR_EL2](#).TGE == 1.

Note

EL2 does not provide traps on debug register accesses through the optional memory-mapped external debug interfaces.

System register accesses to the debug registers might have side-effects. When a System register access is trapped to EL2, no side-effects occur before the exception is taken to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDOSA, bit [10]**When FEAT_DoubleLock is implemented:**

Trap debug OS-related register access. Traps EL1 System register accesses to the powerdown debug registers to EL2, from both Execution states as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x18:
 - [OSLAR_EL1](#), [OSLSR_EL1](#), [OSDLR_EL1](#), and [DBGPRCR_EL1](#).
 - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.
- In AArch32 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x05:
 - [DBGOSLSR](#), [DBGOSLAR](#), [DBGOSDLR](#), and [DBGPRCR](#).
 - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

| TDOSA | Meaning |
|-------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 System register accesses to the powerdown debug registers are trapped to EL2 when EL2 is enabled in the current Security state. |

Note

These registers are not accessible at EL0.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- [MDCR_EL2](#).TDE == 1.
- [HCR_EL2](#).TGE == 1.

System register accesses to the debug registers might have side-effects. When a System register access is trapped to EL2, no side-effects occur before the exception is taken to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Trap debug OS-related register access. Traps EL1 System register accesses to the powerdown debug registers to EL2, from both Execution states as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x18:
 - [OSLAR_EL1](#), [OSLSR_EL1](#), and [DBGPRCR_EL1](#).
 - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.
- In AArch32 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x05:
 - [DBGOSLSR](#), [DBGOSLAR](#), and [DBGPRCR](#).
 - Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

It is IMPLEMENTATION DEFINED whether accesses to [OSDLR_EL1](#) are trapped.

It is IMPLEMENTATION DEFINED whether accesses to [DBGOSDLR](#) are trapped.

| TDOSA | Meaning |
|--------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 System register accesses to the powerdown debug registers are trapped to EL2 when EL2 is enabled in the current Security state. |

Note

These registers are not accessible at EL0.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- [MDCR_EL2.TDE](#) == 1.
- [HCR_EL2.TGE](#) == 1.

Note

EL2 does not provide traps on debug register accesses through the optional memory-mapped external debug interfaces.

System register accesses to the debug registers might have side-effects. When a System register access is trapped to EL2, no side-effects occur before the exception is taken to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDA, bit [9]

Trap Debug Access. Traps EL0 and EL1 System register accesses to debug System registers that are not trapped by MDCR_EL2.TDRA or MDCR_EL2.TDOSA, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2 reported using EC syndrome value 0x18:
 - [MDCCSR_EL0](#), [MDCCINT_EL1](#), [OSDTRRX_EL1](#), [MDSCR_EL1](#), [OSDTRTX_EL1](#), [OSECCR_EL1](#), [DBGBVR<n>_EL1](#), [DBGBCR<n>_EL1](#), [DBGWVR<n>_EL1](#), [DBGWCR<n>_EL1](#), [DBGCLAIMSET_EL1](#), [DBGCLAIMCLR_EL1](#), [DBGAUTHSTATUS_EL1](#).
 - When not in Debug state, [DBGDTR_EL0](#), [DBGDTRRX_EL0](#), [DBGDTRTX_EL0](#).
- In AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x05.
 - [DBGDIDR](#), [DBGDSCRint](#), [DBGDCCINT](#), [DBGWFAR](#), [DBGVCR](#), [DBGDSCRExt](#), [DBGDTRTXext](#), [DBGDTRRXext](#), [DBGBVR<n>](#), [DBGBCR<n>](#), [DBGBXVR<n>](#), [DBGWCR<n>](#), [DBGWVR<n>](#), [DBGCLAIMSET](#), [DBGCLAIMCLR](#), [DBGAUTHSTATUS](#), [DBGDEVID](#), [DBGDEVID1](#), [DBGDEVID2](#), [DBGOSECRR](#).
 - When not in Debug state, [DBGDTRRXint](#) and [DBGDTRTXint](#).
- In AArch32 state, STC accesses to [DBGDTRRXint](#) and LDC accesses to [DBGDTRTXint](#) are trapped to EL2, reported using EC syndrome value 0x06.

| TDA | Meaning |
|------------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL0 or EL1 System register accesses to the debug registers are trapped from both Execution states to EL2 when EL2 is enabled in the current Security state, unless the access generates a higher priority exception. |

Traps of AArch32 accesses to [DBGDTRRXint](#) and [DBGDTRTXint](#) are ignored in Debug state.

Traps of AArch64 accesses to [DBGDTR_EL0](#), [DBGDTRRX_EL0](#), and [DBGDTRTX_EL0](#) are ignored in Debug state.

This field is treated as being 1 for all purposes other than a direct read when one or more of the following are true:

- [MDCR_EL2.TDE](#) == 1
- [HCR_EL2.TGE](#) == 1

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDE, bit [8]

Trap Debug Exceptions. Controls routing of Debug exceptions, and defines the debug target Exception level, EL_D.

| TDE | Meaning |
|------------|--|
| 0b0 | The debug target Exception level is EL1. |
| 0b1 | If EL2 is enabled for the current Effective value of SCR_EL3.NS , the debug target Exception level is EL2, otherwise the debug target Exception level is EL1. The MDCR_EL2.{TDRA, TDOSA, TDA} fields are treated as being 1 for all purposes other than returning the result of a direct read of the register. |

For more information, see 'Routing debug exceptions'.

This field is treated as being 1 for all purposes other than a direct read when [HCR_EL2.TGE](#) == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HPME, bit [7]

When FEAT_PMUv3 is implemented:

[MDCR_EL2.HPMN..(N-1)] event counters enable.

| HPME | Meaning |
|-------------|---|
| 0b0 | Event counters in the range [MDCR_EL2.HPMN.. (PMCR_EL0.N-1)] are disabled. |
| 0b1 | Event counters in the range [MDCR_EL2.HPMN.. (PMCR_EL0.N-1)] are enabled by PMCNTENSET_EL0 . |

If MDCR_EL2.HPMN is less than [PMCR_EL0.N](#), this field affects the operation of event counters in the range [MDCR_EL2.HPMN..[\(PMCR_EL0.N-1\)](#)].

This field does not affect the operation of other event counters.

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPM, bit [6]

When FEAT_PMUv3 is implemented:

Trap Performance Monitors accesses. Traps EL0 and EL1 accesses to all Performance Monitor registers to EL2 when EL2 is enabled in the current Security state, from both Execution states, as follows:

- In AArch64 state, accesses to the following registers are trapped to EL2, reported using EC syndrome value 0x18:
 - [PMCR_EL0](#), [PMCNTENSET_EL0](#), [PMCNTENCLR_EL0](#), [PMOVSCCLR_EL0](#), [PMSWINC_EL0](#), [PMSELR_EL0](#), [PMCEID0_EL0](#), [PMCEID1_EL0](#), [PMCCNTR_EL0](#), [PMXEVTYPER_EL0](#), [PMXEVCNTR_EL0](#), [PMUSERENR_EL0](#), [PMINTENSET_EL1](#), [PMINTENCLR_EL1](#), [PMOVSSET_EL0](#), [PMEVCNTR<n>_EL0](#), [PMEVTYPER<n>_EL0](#), [PMCCFILTR_EL0](#).
 - If FEAT_PMUv3p4 is implemented, [PMMIR_EL1](#)

- In AArch32 state, MRC or MCR accesses to the following registers are trapped to EL2 and reported using EC syndrome value 0x03, MRRC or MCRR accesses are trapped to EL2 and reported using EC syndrome value 0x04:
 - [PMCR](#), [PMCNTENSET](#), [PMCNTENCLR](#), [PMOVSr](#), [PMSWINC](#), [PMSELR](#), [PMCEID0](#), [PMCEID1](#), [PMCCNTR](#), [PMXEVTYPER](#), [PMXEVCNTR](#), [PMUSERENR](#), [PMINTENSET](#), [PMINTENCLR](#), [PMOVSSET](#), [PMEVCNTR<n>](#), [PMEVTYPER<n>](#), [PMCCFILTR](#).
 - If FEAT_PMUv3p1 is implemented, [PMCEID2](#), and [PMCEID3](#).
 - If FEAT_PMUv3p4 is implemented, [PMMIR](#).

| TPM | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL0 and EL1 accesses to all Performance Monitor registers are trapped to EL2 when EL2 is enabled in the current Security state. |

Note

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPMCR, bit [5]

When FEAT_PMUv3 is implemented:

Trap [PMCR_EL0](#) or [PMCR](#) accesses. Traps EL0 and EL1 accesses to EL2, when EL2 is enabled in the current Security state, as follows:

- In AArch64 state, accesses to [PMCR_EL0](#) are trapped to EL2, reported using EC syndrome value 0x18.
- In AArch32 state, accesses to [PMCR](#) are trapped to EL2, reported using EC syndrome value 0x03.

| TPMCR | Meaning |
|-------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL0 and EL1 accesses to the PMCR_EL0 or PMCR are trapped to EL2 when EL2 is enabled in the current Security state, unless it is trapped by PMUSERENR .EN or PMUSERENR_EL0 .EN. |

Note

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPMN, bits [4:0]**When FEAT_PMUv3 is implemented:**

Defines the number of event counters that are accessible from EL3, EL2, EL1, and from EL0 if permitted.

If HPMN is not 0 and is less than [PMCR_ELO.N](#), HPMN divides the Performance Monitors into a first range [0..(HPMN-1)], and a second range [HPMN..([PMCR_ELO.N](#)-1)].

If FEAT_HPMN0 is implemented and this field is 0, all events counters are in the second range and none are in the first range.

If HPMN is equal to [PMCR_ELO.N](#), all event counters are in the first range and none are in the second range.

For an event counter <n> in the first range:

- The counter is accessible from EL1, EL2, and EL3.
- The counter is accessible from EL0 if permitted by [PMUSERENR_ELO](#) or [PMUSERENR](#).
- If FEAT_PMUv3p5 is implemented, [PMCR_ELO.LP](#) or [PMCR.LP](#) determines whether the counter overflow flag is set on unsigned overflow of [PMEVCNTR<n>_ELO\[31:0\]](#) or [PMEVCNTR<n>_ELO\[63:0\]](#).
- [PMCR_ELO.E](#) and [PMCNTENSET_ELO\[n\]](#) enable the operation of event counter n.

For an event counter <n> in the second range:

- The counter is accessible from EL2 and EL3.
- If EL2 is disabled in the current Security state, the event counter is also accessible from EL1, and from EL0 if permitted by [PMUSERENR_ELO](#).
- If FEAT_PMUv3p5 is implemented, MDCR_EL2.HLP determines whether the counter overflow flag is set on unsigned overflow of [PMEVCNTR<n>_ELO\[31:0\]](#) or [PMEVCNTR<n>_ELO\[63:0\]](#).
- MDCR_EL2.HPME and [PMCNTENSET_ELO\[n\]](#) enable the operation of event counter n.

If HPMN is larger than [PMCR_ELO.N](#), or if FEAT_HPMN0 is not implemented, and HPMN is 0, the following CONSTRAINED UNPREDICTABLE behaviors apply:

- The value returned by a direct read of MDCR_EL2.HPMN is UNKNOWN.
- Either:
 - An UNKNOWN number of counters are reserved for EL2 and EL3 use. That is, the PE behaves as if MDCR_EL2.HPMN is set to an UNKNOWN non-zero value less than or equal to [PMCR_ELO.N](#).
 - All counters are reserved for EL2 and EL3 use, meaning no counters are accessible from EL1 and EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to the value in [PMCR_ELO.N](#).

Otherwise:

Reserved, RES0.

Accessing MDCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MDCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MDCR_EL2;
elsif PSTATE.EL == EL3 then
    return MDCR_EL2;

```

MSR MDCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MDCR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MDCR_EL3, Monitor Debug Configuration Register (EL3)

The MDCR_EL3 characteristics are:

Purpose

Provides EL3 configuration options for self-hosted debug and the Performance Monitors Extension.

Configuration

AArch64 System register MDCR_EL3 bits [31:0] can be mapped to AArch32 System register [SDCR\[31:0\]](#), but this is not architecturally mandated.

This register is present only when EL3 is implemented. Otherwise, direct accesses to MDCR_EL3 are UNDEFINED.

Attributes

MDCR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | |
|--------|-------|------|-------|------|------|------|------|------|------|------|-------|-----|-------|------|-------|-------|------|------|----|----|----|----|------|
| 636261 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | E3B1 |
| RES0 | MTPME | TDCC | NSTBE | NSTB | SCCD | ETAD | EPMA | EDAD | TTFR | STES | SPMES | SDD | SPD32 | NSPB | NSPBE | TDOSA | TDAR | RES0 | TP | | | | |
| 313029 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 |

Bits [63:39]

Reserved, RES0.

E3BREC, bit [38]

When FEAT_BRBEv1p1 is implemented:

Branch Record Buffer EL3 Cold Reset Enable. With MDCR_EL3.E3BREW, controls branch recording at EL3.

| E3BREC | Meaning |
|---------------|---|
| 0b0 | When MDCR_EL3.E3BREW == 0: Branch recording at EL3 is disabled. When MDCR_EL3.E3BREW == 1: Branch recording at EL3 is enabled. |
| 0b1 | When MDCR_EL3.E3BREW == 0: Branch recording at EL3 is enabled. When MDCR_EL3.E3BREW == 1: Branch recording at EL3 is disabled. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

E3BREW, bit [37]**When FEAT_BRBEv1p1 is implemented:**

Branch Record Buffer EL3 Warm Reset Enable. With MDCR_EL3.E3BREC, controls branch recording at EL3.

For a description of the values derived by evaluating MDCR_EL3.E3BREC and MDCR_EL3.E3BREW together, see MDCR_EL3.E3BREC.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

EnPMSN, bit [36]**When FEAT_SPEv1p2 is implemented:**

Trap accesses to [PMSNEVFR_EL1](#). Controls access to Statistical Profiling [PMSNEVFR_EL1](#) System register from EL2 and EL1.

| EnPMSN | Meaning |
|--------|---|
| 0b0 | Accesses to PMSNEVFR_EL1 at EL2 and EL1 generate a Trap exception to EL3. |
| 0b1 | Do not trap PMSNEVFR_EL1 to EL3. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MPMX, bit [35]**When FEAT_PMUv3p7 is implemented:**

Monitor Performance Monitors Extended control. In conjunction with MDCR_EL3.SPME, controls when event counters are enabled at EL3 and in other Secure Exception levels.

| MPMX | Meaning |
|------|--|
| 0b0 | Event counting and PMCCNTR_EL0 are not affected by this mechanism. |
| 0b1 | Event counting by some or all event counters is prohibited at EL3. If PMCR_EL0 .DP is 1, PMCCNTR_EL0 is disabled at EL3. Otherwise, PMCCNTR_EL0 is not affected by this mechanism. |

If EL2 is implemented, MDCR_EL3.SPME == 1, and [MDCR_EL2](#).HPMN is less than [PMCR_EL0](#).N then all the following are true:

- If [MDCR_EL2](#).HPMN is not 0, this field affects the operation of event counters in the range [0 .. ([MDCR_EL2](#).HPMN-1)] at EL3.
- This field does not affect the operation of other event counters.
- If [PMCR_EL0](#).DP is 1, this field affects the operation of [PMCCNTR_EL0](#) at EL3.

The operation of this field applies even when EL2 is disabled in the current Security state.

If EL2 is not implemented, MDCR_EL3.SPME == 0, or [MDCR_EL2](#).HPMN is equal to [PMCR_EL0](#).N then this field affects the operation of all event counters at EL3, and if [PMCR_EL0](#).DP is 1, the operation of [PMCCNTR_EL0](#) at EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

MCCD, bit [34]

When FEAT_PMUv3p7 is implemented:

Monitor Cycle Counter Disable. Prohibits the Cycle Counter, [PMCCNTR_EL0](#), from counting at EL3.

| MCCD | Meaning |
|------|--|
| 0b0 | Cycle counting by PMCCNTR_EL0 is not affected by this mechanism. |
| 0b1 | Cycle counting by PMCCNTR_EL0 is prohibited at EL3. |

This field does not affect the CPU_CYCLES event or any other event that counts cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SBRBE, bits [33:32]

When FEAT_BRBE is implemented:

Secure Branch Record Buffer Enable. Controls branch recording by the BRBE, and access to BRBE registers and instructions at EL2 and EL1.

| SBRBE | Meaning |
|-------|---|
| 0b00 | Direct accesses to BRBE registers and instructions, except when in EL3, generate a Trap exception to EL3. EL0, EL1, and EL2 are prohibited regions. |
| 0b01 | Direct accesses to BRBE registers and instructions in Secure state, except when in EL3, generate a Trap exception to EL3. EL0, EL1, and EL2 in Secure state are prohibited regions. This control does not cause any direct accesses to BRBE registers when not in Secure state to be trapped, and does not cause any Exception levels when not in Secure state to be a prohibited region. |
| 0b10 | Direct accesses to BRBE registers and instructions, except when in EL3, generate a Trap exception to EL3. This control does not cause any Exception levels to be prohibited regions. |
| 0b11 | This control does not cause any direct accesses to BRBE registers or instruction to be trapped, and does not cause any Exception levels to be a prohibited region. |

The Branch Record Buffer registers trapped by this control are: [BRBCR_EL1](#), [BRBCR_EL2](#), [BRBCR_EL12](#), [BRBFCR_EL1](#), [BRBIDR0_EL1](#), [BRBINF<n>_EL1](#), [BRBINFINJ_EL1](#), [BRBSRC<n>_EL1](#), [BRBSRCINJ_EL1](#), [BRBTGT<n>_EL1](#), [BRBTGTINJ_EL1](#), and [BRBTS_EL1](#).

The Branch Record Buffer instructions trapped by this control are:

- [BRB IALL](#).
- [BRB INJ](#).

Note

If FEAT_BRBEv1p1 is not implemented, EL3 is a prohibited region.

If EL3 is not implemented then the Effective value of this field is 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [31:29]

Reserved, RES0.

MTPME, bit [28]

When FEAT_MTPMU is implemented:

Multi-threaded PMU Enable. Enables use of the [PMEVTYPER<n>_ELO](#).MT bits.

| MTPME | Meaning |
|-------|--|
| 0b0 | FEAT_MTPMU is disabled. The Effective value of PMEVTYPER<n>_ELO .MT is zero. |
| 0b1 | PMEVTYPER<n>_ELO .MT bits not affected by this field. |

If FEAT_MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is IMPLEMENTATION DEFINED whether the PE behaves as if this field is 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to 1.

Otherwise:

Reserved, RES0.

TDCC, bit [27]

When FEAT_FGT is implemented:

Trap DCC. Traps use of the Debug Comms Channel at EL2, EL1, and EL0 to EL3.

| TDCC | Meaning |
|------|---|
| 0b0 | This control does not cause any register accesses to be trapped. |
| 0b1 | Accesses to the DCC registers at EL2, EL1, and EL0 generate a Trap exception to EL3, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state. |

The DCC registers trapped by this control are:

AArch64: [OSDTRRX_EL1](#), [OSDTRTX_EL1](#), [MDCCSR_EL0](#), [MDCCINT_EL1](#), and, when the PE is in Non-debug state, [DBGDTR_EL0](#), [DBGDTRRX_EL0](#), and [DBGDTRTX_EL0](#).

AArch32: [DBGDTRRXext](#), [DBGDTRTXext](#), [DBGDSCRint](#), [DBGDCCINT](#), and, when the PE is in Non-debug state, [DBGDTRRXint](#) and [DBGDTRTXint](#).

The traps are reported with EC syndrome value:

- 0x05 for trapped AArch32 MRC and MCR accesses with coproc == 0b1110.
- 0x06 for trapped AArch32 LDC to [DBGDTRTXint](#) and STC from [DBGDTRRXint](#).
- 0x18 for trapped AArch64 MRS and MSR accesses.

When the PE is in Debug state, MDCR_EL3.TDCC does not trap any accesses to:

AArch64: [DBGDTR_EL0](#), [DBGDTRRX_EL0](#), and [DBGDTRTX_EL0](#).

AArch32: [DBGDTRRXint](#) and [DBGDTRTXint](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSTBE, bit [26]

When FEAT_TRBE is implemented and FEAT_RME is implemented:

Non-secure Trace Buffer Extended. Together with MDCR_EL3.NSTB, controls the owning translation regime and accesses to Trace Buffer control registers from EL2 and EL1.

For a description of the values derived by evaluating NSTB and NSTBE together, see MDCR_EL3.NSTB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSTB, bits [25:24]

When FEAT_TRBE is implemented and FEAT_RME is implemented:

Non-secure Trace Buffer. Together with MDCR_EL3.NSTBE, controls the owning translation regime and accesses to Trace Buffer control registers from EL2 and EL1.

| NSTB | Meaning |
|-------------|---|
| 0b00 | When MDCR_EL3.NSTBE == 0b0: Trace Buffer owning security state is Secure state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Non-secure and Realm state. Accesses to Trace Buffer control registers at EL2 and EL1 generate Trap exceptions to EL3. When MDCR_EL3.NSTBE == 0b1: Reserved. |
| 0b01 | When MDCR_EL3.NSTBE == 0b0: Trace Buffer owning security state is Secure state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Non-secure and Realm state. Accesses to Trace Buffer control registers at EL2 and EL1 in Non-secure and Realm state generate Trap exceptions to EL3. When MDCR_EL3.NSTBE == 0b1: Reserved. |
| 0b10 | When MDCR_EL3.NSTBE == 0b0: Trace Buffer owning security state is Non-secure state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Secure and Realm state. Accesses to Trace Buffer control registers at EL2 and EL1 generate Trap exceptions to EL3. When MDCR_EL3.NSTBE == 0b1: Trace Buffer owning security state is Realm state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Non-secure and Secure state. Accesses to Trace Buffer control registers at EL2 and EL1 generate Trap exceptions to EL3. |
| 0b11 | When MDCR_EL3.NSTBE == 0b0: Trace Buffer owning security state is Non-secure state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Secure and Realm state. Accesses to Trace Buffer control registers at EL2 and EL1 in Secure and Realm state generate Trap exceptions to EL3. When MDCR_EL3.NSTBE == 0b1: Trace Buffer owning security state is Realm state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Non-secure and Secure state. Accesses to Trace Buffer control registers at EL2 and EL1 in Non-secure and Secure state generate Trap exceptions to EL3. |

The Trace Buffer control registers trapped by this control are: [TRBBASER_EL1](#), [TRBLIMITR_EL1](#), [TRBMAR_EL1](#), [TRBPTR_EL1](#), [TRBSR_EL1](#), and [TRBTRG_EL1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_TRBE is implemented and FEAT_RME is not implemented:

Non-secure Trace Buffer. Controls the owning translation regime and accesses to Trace Buffer control registers from EL2 and EL1.

| NSTB | Meaning |
|------|--|
| 0b00 | Trace Buffer owning security state is Secure state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Non-secure state. Accesses to Trace Buffer control registers at EL2 and EL1 generate Trap exceptions to EL3. |
| 0b01 | Trace Buffer owning security state is Secure state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Non-secure state. Accesses to Trace Buffer control registers at EL2 and EL1 in Non-secure state generate Trap exceptions to EL3. |
| 0b10 | Trace Buffer owning security state is Non-secure state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Secure state. Accesses to Trace Buffer control registers at EL2 and EL1 generate Trap exceptions to EL3. |
| 0b11 | Trace Buffer owning security state is Non-secure state. If TraceBufferEnabled() == TRUE, tracing is prohibited in Secure state. Accesses to Trace Buffer control registers at EL2 and EL1 in Secure state generate Trap exceptions to EL3. |

The Trace Buffer control registers trapped by this control are: [TRBBASER_EL1](#), [TRBLIMITR_EL1](#), [TRBMAR_EL1](#), [TRBPTR_EL1](#), [TRBSR_EL1](#), and [TRBTRG_EL1](#).

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 1, then the Effective value of this field is 0b11.

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 0, then the Effective value of this field is 0b01.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SCCD, bit [23]

When FEAT_PMUv3p5 is implemented:

Secure Cycle Counter Disable. Prohibits [PMCCNTR_EL0](#) from counting in Secure state.

| SCCD | Meaning |
|------|--|
| 0b0 | Cycle counting by PMCCNTR_EL0 is not affected by this mechanism. |
| 0b1 | Cycle counting by PMCCNTR_EL0 is prohibited in Secure state. |

This field does not affect the CPU_CYCLES event or any other event that counts cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

ETAD, bit [22]

When FEAT_RME is implemented, external debugger access to the PE Trace Unit registers is implemented and FEAT_TRBE is implemented:

External Trace Access Disable. Together with MDCR_EL3.ETADE, controls access to PE Trace Unit registers by an external debugger.

| ETADE | ETAD | Meaning |
|-------|------|--|
| 0b0 | 0b0 | Access to PE Trace Unit registers by an external debugger is permitted. |
| 0b0 | 0b1 | Root and Secure access to PE Trace Unit registers by an external debugger is permitted. Realm and Non-secure access to PE Trace Unit registers by an external debugger is not permitted. |
| 0b1 | 0b0 | Root and Realm access to PE Trace Unit registers by an external debugger is permitted. Secure and Non-secure access to PE Trace Unit registers by an external debugger is not permitted. |
| 0b1 | 0b1 | Root access to PE Trace Unit registers by an external debugger is permitted. Secure, Non-secure, and Realm access to PE Trace Unit registers by an external debugger is not permitted. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

When external debugger access to the PE Trace Unit registers is implemented and FEAT_TRBE is implemented:

External Trace Access Disable. Controls Non-secure access to PE Trace Unit registers by an external debugger.

| ETAD | Meaning |
|------|--|
| 0b0 | Non-secure accesses from an external debugger to PE Trace Unit are allowed. |
| 0b1 | Non-secure accesses from an external debugger to some PE Trace Unit registers are prohibited. See individual registers for the effect of this field. |

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

EPMADE, bit [21]

When FEAT_RME is implemented, FEAT_PMUV3 is implemented and the Performance Monitors Extension supports external debug interface accesses:

External Performance Monitors Access Disable. Together with MDCR_EL3.EPMADE, controls access to Performance Monitor registers by an external debugger.

| EPMADE | EPMAD | Meaning |
|--------|-------|--|
| 0b0 | 0b0 | Access to Performance Monitor registers by an external debugger is permitted. |
| 0b0 | 0b1 | Root and Secure access to Performance Monitor registers by an external debugger is permitted. Realm and Non-secure access to Performance Monitor registers by an external debugger is not permitted. |
| 0b1 | 0b0 | Root and Realm access to Performance Monitor registers by an external debugger is permitted. Secure and Non-secure access to Performance Monitor registers by an external debugger is not permitted. |
| 0b1 | 0b1 | Root access to Performance Monitor registers by an external debugger is permitted. Secure, Non-secure, and Realm access to Performance Monitor registers by an external debugger is not permitted. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

When FEAT_Debugv8p4 is implemented, FEAT_PMUv3 is implemented and the Performance Monitors Extension supports external debug interface accesses:

External Performance Monitors Non-secure Access Disable. Controls Non-secure access to Performance Monitor registers by an external debugger.

| EPMAD | Meaning |
|-------|---|
| 0b0 | Non-secure access to Performance Monitor registers from external debugger is permitted. |
| 0b1 | Non-secure access to Performance Monitor registers from external debugger is not permitted. |

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 0b0, then the Effective value of this bit is 0b1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

When FEAT_PMUv3 is implemented and the Performance Monitors Extension supports external debug interface accesses:

External Performance Monitors Access Disable. Controls access to Performance Monitor registers by an external debugger.

| EPMAD | Meaning |
|-------|--|
| 0b0 | Access to Performance Monitor registers from external debugger is permitted. |
| 0b1 | Access to Performance Monitor registers from external debugger is not permitted, unless overridden by the IMPLEMENTATION DEFINED authentication interface. |

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 0b0, then the Effective value of this bit is 0b1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

EDAD, bit [20]**When FEAT_RME is implemented:**

External Debug Access Disable. Together with MDCR_EL3.EDADE, controls access to breakpoint registers, watchpoint registers, and [OSLAR_EL1](#) by an external debugger.

| EDADE | EDAD | Meaning |
|-------|------|--|
| 0b0 | 0b0 | Access to Debug registers by an external debugger is permitted. |
| 0b0 | 0b1 | Root and Secure access to Debug registers by an external debugger is permitted. Realm and Non-secure access to Debug registers by an external debugger is not permitted. |
| 0b1 | 0b0 | Root and Realm access to Debug registers by an external debugger is permitted. Secure and Non-secure access to Debug registers by an external debugger is not permitted. |
| 0b1 | 0b1 | Root access to Debug registers by an external debugger is permitted. Secure, Non-secure, and Realm access to Debug registers by an external debugger is not permitted. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

When FEAT_Debugv8p4 is implemented:

External Debug Non-secure Access Disable. Controls Non-secure access to breakpoint, watchpoint, and [OSLAR_EL1](#) registers by an external debugger.

| EDAD | Meaning |
|------|--|
| 0b0 | Non-secure access to debug registers from external debugger is permitted. |
| 0b1 | Non-secure access to breakpoint and watchpoint registers, and OSLAR_EL1 from external debugger is not permitted. |

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 0b0, then the Effective value of this field is 0b1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

When FEAT_Debugv8p2 is implemented:

External Debug Access Disable. Controls access to breakpoint, watchpoint, and [OSLAR_EL1](#) registers by an external debugger.

| EDAD | Meaning |
|------|--|
| 0b0 | Access to debug registers, and to OSLAR_EL1 from external debugger is permitted. |
| 0b1 | Access to breakpoint and watchpoint registers, and to OSLAR_EL1 from external debugger is not permitted, unless overridden by the IMPLEMENTATION DEFINED authentication interface. |

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 0b0, then the Effective value of this field is 0b1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

External Debug Access disable. Controls access to breakpoint, watchpoint, and optionally [OSLAR_EL1](#) registers by an external debugger.

| EDAD | Meaning |
|------|--|
| 0b0 | Access to debug registers from external debugger is permitted. |
| 0b1 | Access to breakpoint and watchpoint registers from an external debugger is not permitted, unless overridden by the IMPLEMENTATION DEFINED authentication interface. It is IMPLEMENTATION DEFINED whether access to the OSLAR_EL1 register from an external debugger is permitted or not permitted. |

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 0b0, then the Effective value of this field is 0b1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

TTRF, bit [19]**When FEAT_TRF is implemented:**

Trap Trace Filter controls. Traps use of the Trace Filter control registers at EL2 and EL1 to EL3.

The Trace Filter registers trapped by this control are:

- [TRFCR_EL2](#), TRFCR_EL12, [TRFCR_EL1](#), reported using EC syndrome value 0x18.
- [HTRFCR](#) and [TRFCR](#), reported using EC syndrome value 0x03.

| TTRF | Meaning |
|------|--|
| 0b0 | Accesses to Trace Filter registers at EL2 and EL1 are not affected by this bit. |
| 0b1 | Accesses to Trace Filter registers at EL2 and EL1 generate a Trap exception to EL3, unless the access generates a higher priority exception. |

Otherwise:

Reserved, RES0.

STE, bit [18]**When FEAT_TRF is implemented:**

Secure Trace enable. Enables tracing in Secure state.

| STE | Meaning |
|-----|--|
| 0b0 | Trace prohibited in Secure state unless overridden by the IMPLEMENTATION DEFINED authentication interface. |
| 0b1 | Trace in Secure state is not affected by this bit. |

This bit also controls the level of authentication required by an external debugger to enable external tracing. See 'Register controls to enable self-hosted trace'.

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 0b0, the Effective value of this bit is 0b1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SPME, bit [17]

When FEAT_PMUv3 is implemented and FEAT_PMUv3p7 is implemented:

Secure Performance Monitors Enable. Controls event counting in Secure state and EL3.

| SPME | Meaning |
|------|--|
| 0b0 | When MDCR_EL3.MPMX == 0: Event counting is prohibited in Secure state. If PMCR_EL0.DP is 1, PMCCNTR_EL0 is disabled in Secure state. Otherwise, PMCCNTR_EL0 is not affected by this mechanism. |
| 0b1 | When MDCR_EL3.MPMX == 0: Event counting and PMCCNTR_EL0 are not affected by this mechanism. |

When MDCR_EL3.MPMX is 0, this field affects the operation of all event counters in Secure state, and if [PMCR_EL0.DP](#) is 1, the operation of [PMCCNTR_EL0](#) in Secure state.

When MDCR_EL3.MPMX is 1, this field affects the operation of event counters at EL3 only, and if [PMCR_EL0.DP](#) is 1, the operation of [PMCCNTR_EL0](#) at EL3 only. See MDCR_EL3.MPMX for more information.

When [PMCR_EL0.DP](#) is 0, [PMCCNTR_EL0](#) is not affected by this field.

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

When FEAT_PMUv3 is implemented and FEAT_Debugv8p2 is implemented:

Secure Performance Monitors Enable. Controls event counting in Secure state.

| SPME | Meaning |
|------|---|
| 0b0 | Event counting is prohibited in Secure state. If PMCR_EL0.DP is 1, PMCCNTR_EL0 is disabled in Secure state. Otherwise, PMCCNTR_EL0 is not affected by this mechanism. |
| 0b1 | Event counting and PMCCNTR_EL0 are not affected by this mechanism. |

This field affects the operation of all event counters in Secure state, and if [PMCR_EL0.DP](#) is 1, the operation of [PMCCNTR_EL0](#) in Secure state. When [PMCR_EL0.DP](#) is 0, [PMCCNTR_EL0](#) is not affected by this field.

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

When FEAT_PMUv3 is implemented:

Secure Performance Monitors Enable. Controls event counting in Secure state.

| SPME | Meaning |
|------|---|
| 0b0 | If <code>ExternalSecureNoninvasiveDebugEnabled()</code> is FALSE, event counting is prohibited in Secure state, and if PMCR_EL0.DP is 1, PMCCNTR_EL0 is disabled in Secure state. |
| 0b1 | Event counting and PMCCNTR_EL0 are not affected by this mechanism. |

If `ExternalSecureNoninvasiveDebugEnabled()` is TRUE, the event counters and [PMCCNTR_EL0](#) are not affected by this field.

Otherwise, this field affects the operation of all event counters in Secure state, and if [PMCR_EL0](#).DP is 1, the operation of [PMCCNTR_EL0](#) in Secure state. When [PMCR_EL0](#).DP is 0, [PMCCNTR_EL0](#) is not affected by this field.

If EL3 is not implemented and the Effective value of [SCR_EL3](#).NS is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SDD, bit [16]

AArch64 Secure Self-hosted invasive debug disable. Disables Software debug exceptions in Secure state, other than Breakpoint Instruction exceptions.

| SDD | Meaning |
|-----|---|
| 0b0 | Debug exceptions in Secure state are not affected by this bit. |
| 0b1 | Debug exceptions, other than Breakpoint Instruction exceptions, are disabled from all Exception levels in Secure state. |

The SDD bit is ignored unless both of the following are true:

- The PE is in Secure state.
- The Effective value of [SCR_EL3](#).RW is 0b1.

If Secure EL2 is implemented and enabled, and Secure EL1 is using AArch32, then:

- If debug exceptions from Secure EL1 are enabled, debug exceptions from Secure EL0 are also enabled.
- Otherwise, debug exceptions from Secure EL0 are enabled only if the value of [SDER32_EL3](#).SUIDEN is 0b1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SPD32, bits [15:14]

When EL1 is capable of using AArch32:

AArch32 Secure self-hosted privileged debug. Enables or disables debug exceptions from Secure EL1 using AArch32, other than Breakpoint Instruction exceptions.

| SPD32 | Meaning |
|-------|---|
| 0b00 | Legacy mode. Debug exceptions from Secure EL1 are enabled by the IMPLEMENTATION DEFINED authentication interface. |
| 0b10 | Secure privileged debug disabled. Debug exceptions from Secure EL1 are disabled. |
| 0b11 | Secure privileged debug enabled. Debug exceptions from Secure EL1 are enabled. |

Other values are reserved, and have the CONSTRAINED UNPREDICTABLE behavior that they must have the same behavior as 0b00. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

This field has no effect on Breakpoint Instruction exceptions. These are always enabled.

This field is ignored unless both of the following are true:

- The PE is in Secure state.

- The Effective value of [SCR_EL3.RW](#) is 0b0.

If Secure EL1 is using AArch32, then:

- If debug exceptions from Secure EL1 are enabled, then debug exceptions from Secure EL0 are also enabled.
- Otherwise, debug exceptions from Secure EL0 are enabled only if the value of [SDER32_EL3.SUIDEN](#) is 0b1.

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 0b0, then the Effective value of this field is 0b11.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSPB, bits [13:12]

When FEAT_SPE is implemented and FEAT_RME is implemented:

Non-secure Profiling Buffer. Together with MDCR_EL3.NSPBE, controls the owning translation regime and accesses to Statistical Profiling and Profiling Buffer control registers.

| NSPB | Meaning |
|------|--|
| 0b00 | When MDCR_EL3.NSPBE == 0b0: Profiling Buffer uses Secure Virtual Addresses. Statistical Profiling enabled in Secure state and disabled in Non-secure and Realm state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in all Security states generate Trap exceptions to EL3. When MDCR_EL3.NSPBE == 0b1: Reserved. |
| 0b01 | When MDCR_EL3.NSPBE == 0b0: Profiling Buffer uses Secure Virtual Addresses. Statistical Profiling enabled in Secure state and disabled in Non-secure and Realm state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in Non-secure and Realm states generate Trap exceptions to EL3. When MDCR_EL3.NSPBE == 0b1: Reserved. |
| 0b10 | When MDCR_EL3.NSPBE == 0b0: Profiling Buffer uses Non-secure Virtual Addresses. Statistical Profiling enabled in Non-secure state and disabled in Secure and Realm state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in all Security states generate Trap exceptions to EL3. When MDCR_EL3.NSPBE == 0b1: Profiling Buffer uses Realm Virtual Addresses. Statistical Profiling enabled in Realm state and disabled in Non-secure and Secure state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in all Security states generate Trap exceptions to EL3. |
| 0b11 | When MDCR_EL3.NSPBE == 0b0: Profiling Buffer uses Non-secure Virtual Addresses. Statistical Profiling enabled in Non-secure state and disabled in Secure and Realm state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in Secure and Realm states generate Trap exceptions to EL3. When MDCR_EL3.NSPBE == 0b1: Profiling Buffer uses Realm Virtual Addresses. Statistical Profiling enabled in Realm state and disabled in Non-secure and Secure state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in Non-secure and Secure states generate Trap exceptions to EL3. |

The Statistical Profiling and Profiling Buffer control registers trapped by this control are:

- [PMBLIMITR_EL1](#), [PMBPTR_EL1](#), [PMBSR_EL1](#), [PMSCR_EL1](#), [PMSCR_EL2](#), [PMSCR_EL12](#), [PMSEVFR_EL1](#), [PMSFCR_EL1](#), [PMSICR_EL1](#), [PMSIDR_EL1](#), [PMSIRR_EL1](#), and [PMSLATFR_EL1](#).
- If FEAT_SPEv1p2 is implemented, [PMSNEVFR_EL1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SPE is implemented and FEAT_RME is not implemented:

Non-secure Profiling Buffer. Controls the owning translation regime and accesses to Statistical Profiling and Profiling Buffer control registers.

| NSPB | Meaning |
|------|--|
| 0b00 | Profiling Buffer uses Secure Virtual Addresses. Statistical Profiling enabled in Secure state and disabled in Non-secure state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in Non-secure and Secure states generate Trap exceptions to EL3. |
| 0b01 | Profiling Buffer uses Secure Virtual Addresses. Statistical Profiling enabled in Secure state and disabled in Non-secure state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in Non-secure state generate Trap exceptions to EL3. |
| 0b10 | Profiling Buffer uses Non-secure Virtual Addresses. Statistical Profiling enabled in Non-secure state and disabled in Secure state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in Non-secure and Secure states generate Trap exceptions to EL3. |
| 0b11 | Profiling Buffer uses Non-secure Virtual Addresses. Statistical Profiling enabled in Non-secure state and disabled in Secure state. Accesses to Statistical Profiling and Profiling Buffer control registers at EL2 and EL1 in Secure state generate Trap exceptions to EL3. |

The Statistical Profiling and Profiling Buffer control registers trapped by this control are:

- [PMBLIMITR_EL1](#), [PMBPTR_EL1](#), [PMBSR_EL1](#), [PMSCR_EL1](#), [PMSCR_EL2](#), [PMSCR_EL12](#), [PMSEVFR_EL1](#), [PMSFCR_EL1](#), [PMSICR_EL1](#), [PMSIDR_EL1](#), [PMSIRR_EL1](#), and [PMSLATFR_EL1](#).
- If FEAT_SPEv1p2 is implemented, [PMSNEVFR_EL1](#).

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 1, then the Effective value of this field is 0b11.

If EL3 is not implemented and the Effective value of [SCR_EL3.NS](#) is 0, then the Effective value of this field is 0b01.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSPBE, bit [11]

When FEAT_RME is implemented:

Non-secure Profiling Buffer Extended. Together with MDCR_EL3.NSPB, controls the owning translation regime and accesses to Statistical Profiling and Profiling Buffer control registers.

For a description of the values derived by evaluating NSPB and NSPBE together, see MDCR_EL3.NSPB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TDOSA, bit [10]**When FEAT_DoubleLock is implemented:**

Trap debug OS-related register access. Traps EL2 and EL1 System register accesses to the powerdown debug registers to EL3.

Accesses to the registers are trapped as follows:

- Accesses from AArch64 state, [OSLAR_EL1](#), [OSLSR_EL1](#), [OSDLR_EL1](#), [DBGPRCR_EL1](#), and any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit, are trapped to EL3 and reported using EC syndrome value 0x18.
- Accesses using MCR or MRC to [DBGOSLAR](#), [DBGOSLSR](#), [DBGOSDLR](#), and [DBGPRCR](#), are trapped to EL3 and reported using EC syndrome value 0x05.
- Accesses to any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

| TDOSA | Meaning |
|-------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL2 and EL1 System register accesses to the powerdown debug registers are trapped to EL3, unless it is trapped by HDCR.TDOSA or MDCR_EL2.TDOSA . |

Note

The powerdown debug registers are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Trap debug OS-related register access. Traps EL2 and EL1 System register accesses to the powerdown debug registers to EL3.

The following registers are affected by this trap:

- AArch64: [OSLAR_EL1](#), [OSLSR_EL1](#), and [DBGPRCR_EL1](#).
- AArch32: [DBGOSLAR](#), [DBGOSLSR](#), and [DBGPRCR](#).
- AArch64 and AArch32: Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.
- It is IMPLEMENTATION DEFINED whether accesses to [OSDLR_EL1](#) and [DBGOSDLR](#) are trapped.

| TDOSA | Meaning |
|-------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL2 and EL1 System register accesses to the powerdown debug registers are trapped to EL3, unless it is trapped by HDCR.TDOSA or MDCR_EL2.TDOSA . |

Note

The powerdown debug registers are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDA, bit [9]

Trap Debug Access. Traps EL2, EL1, and EL0 System register accesses to those debug System registers that cannot be trapped using the MDCR_EL3.TDOSA field.

Accesses to the debug registers are trapped as follows:

- In AArch64 state, the following registers are trapped to EL3 and reported using EC syndrome value 0x18:
 - [DBGBVR<n>_EL1](#), [DBGBCR<n>_EL1](#), [DBGWVR<n>_EL1](#), [DBGWCR<n>_EL1](#), [DBGCLAIMSET_EL1](#), [DBGCLAIMCLR_EL1](#), [DBGAUTHSTATUS_EL1](#), [DBGVCR32_EL2](#).
 - AArch64: [MDCR_EL2](#), [MDRAR_EL1](#), [MDCCSR_EL0](#), [MDCCINT_EL1](#), [MDSCR_EL1](#), [OSDTRRX_EL1](#), [OSDTRTX_EL1](#), [OSECCR_EL1](#).
- In AArch32 state, [SDER](#) is trapped to EL3 and reported using EC syndrome value 0x03.
- In AArch32 state, accesses using MCR or MRC to the following registers are reported using EC syndrome value 0x05, accesses using MCRR or MRRC are reported using EC syndrome value 0x0C:
 - [HDCR](#), [DBGDRAR](#), [DBGDSAR](#), [DBGDIDR](#), [DBGDCCINT](#), [DBGWFAR](#), [DBGVCR](#), [DBGBVR<n>](#), [DBGBCR<n>](#), [DBGBXVR<n>](#), [DBGWCR<n>](#), [DBGWVR<n>](#).
 - [DBGCLAIMSET](#), [DBGCLAIMCLR](#), [DBGAUTHSTATUS](#), [DBGDEVID](#), [DBGDEVID1](#), [DBGDEVID2](#), [DBGOSECCR](#).
- In AArch32 state, STC accesses to [DBGDTRRXint](#) and LDC accesses to [DBGDTRTXint](#) are reported using EC syndrome value 0x06.
- When not in Debug state, the following registers are also trapped to EL3:
 - AArch64 accesses to [DBGDTR_EL0](#), [DBGDTRRX_EL0](#), and [DBGDTRTX_EL0](#), reported using EC syndrome value 0x18.
 - AArch32 accesses using MCR or MRC to [DBGDTRRXint](#) and [DBGDTRTXint](#), reported using EC syndrome value 0x05.

| TDA | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL0, EL1, and EL2 accesses to the debug registers, other than the registers that can be trapped by MDCR_EL3.TDOSA, are trapped to EL3, from any Security state and both Execution states, unless it is trapped by DBGDSCRExt .UDCCdis, MDSCR_EL1 .TDCC, HDCR .TDA or MDCR_EL2 .TDA. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:7]

Reserved, RES0.

TPM, bit [6]

When FEAT_PMUv3 is implemented:

Trap Performance Monitor register accesses. Accesses to all Performance Monitor registers from EL0, EL1, and EL2 to EL3, from any Security state and both Execution states are trapped as follows:

- In AArch64 state, accesses to the following registers are trapped to EL3 and are reported using EC syndrome value 0x18:
 - [PMCR_EL0](#), [PMCNTENSET_EL0](#), [PMCNTENCLR_EL0](#), [PMOVSLR_EL0](#), [PMSWINC_EL0](#), [PMSELR_EL0](#), [PMCEID0_EL0](#), [PMCEID1_EL0](#), [PMCCNTR_EL0](#), [PMXEVTYPER_EL0](#), [PMXEVCNTR_EL0](#), [PMUSERENR_EL0](#), [PMINTENSET_EL1](#), [PMINTENCLR_EL1](#), [PMOVSET_EL0](#), [PMEVCNTR<n>_EL0](#), [PMEVTYPER<n>_EL0](#), [PMCCFILTR_EL0](#).
 - If FEAT_PMUv3p4 is implemented, [PMMIR_EL1](#).
- In AArch32 state, accesses using MCR or MRC to the following registers are reported using EC syndrome value 0x03, accesses using MCRR or MRRC are reported using EC syndrome value 0x04:
 - [PMCR](#), [PMCNTENSET](#), [PMCNTENCLR](#), [PMOVSR](#), [PMSWINC](#), [PMSELR](#), [PMCEID0](#), [PMCEID1](#), [PMCCNTR](#), [PMXEVTYPER](#), [PMXEVCNTR](#), [PMUSERENR](#), [PMINTENSET](#), [PMINTENCLR](#), [PMOVSET](#), [PMEVCNTR<n>](#), [PMEVTYPER<n>](#), [PMCCFILTR](#).
 - If FEAT_PMUv3p1 is implemented, [PMCEID2](#), and [PMCEID3](#).
 - If FEAT_PMUv3p4 is implemented, [PMMIR](#).

| TPM | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL2, EL1, and EL0 System register accesses to all Performance Monitor registers are trapped to EL3, unless it is trapped by HDCR .TPM or MDCR_EL2 .TPM. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [5]

Reserved, RES0.

EDADE, bit [4]

When FEAT_RME is implemented:

External Debug Access Disable Extended. Together with MDCR_EL3.EDAD, controls access to breakpoint registers, watchpoint registers, and [OSLAR_EL1](#) by an external debugger.

For a description of the values derived by evaluating EDAD and EDADE together, see MDCR_EL3.EDAD.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

ETADE, bit [3]

When FEAT_RME is implemented, external debugger access to the PE Trace Unit registers is implemented and FEAT_TRBE is implemented:

External Trace Access Disable Extended. Together with MDCR_EL3.ETAD, controls access to PE Trace Unit registers by an external debugger.

For a description of the values derived by evaluating ETAD and ETADE together, see MDCR_EL3.ETAD.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

EPMAD, bit [2]

When FEAT_RME is implemented, FEAT_PMuV3 is implemented and the Performance Monitors Extension supports external debug interface accesses:

External Performance Monitors Access Disable Extended. Together with MDCR_EL3.EPMAD, controls access to Performance Monitor registers by an external debugger.

For a description of the values derived by evaluating EPMAD and EPMAD together, see MDCR_EL3.EPMAD.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [1]

Reserved, RES0.

RLTE, bit [0]

When FEAT_RME is implemented and FEAT_TRF is implemented:

Realm Trace enable. Enables tracing in Realm state.

| RLTE | Meaning |
|------|--|
| 0b0 | Trace prohibited in Realm state, unless overridden by the IMPLEMENTATION DEFINED authentication interface. |
| 0b1 | Trace in Realm state is not affected by this bit. |

This bit also controls the level of authentication that is required by an external debugger to enable external tracing.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Accessing MDCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MDCR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return MDCR_EL3;

```

MSR MDCR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0011 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    MDCR_EL3 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MDRAR_EL1, Monitor Debug ROM Address Register

The MDRAR_EL1 characteristics are:

Purpose

Defines the base physical address of a 4KB-aligned memory-mapped debug component, usually a ROM table that locates and describes the memory-mapped debug components in the system. Armv8 deprecates any use of this register.

Configuration

AArch64 System register MDRAR_EL1 bits [63:0] are architecturally mapped to AArch32 System register [DBGDRAR\[63:0\]](#).

Attributes

MDRAR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | ROMADDR | | | | | | | | | | | | | | | | | | | |
| ROMADDR | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | Valid | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:52]

Reserved, RES0.

ROMADDR, bits [51:12]

ROMADDR encoding when FEAT_LPA is implemented and MDRAR_EL1.Valid != 0b00

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ROMADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

ROMADDR, bits [39:0]

Bits [51:12] of the ROM table physical address.

Bits [11:0] of the ROM table physical address are zero.

For implementations with fewer than 52 physical address bits, the corresponding upper bits of this field are RES0.

In an implementation that includes EL3, ROMADDR is an address in Non-secure memory. It is IMPLEMENTATION DEFINED whether the ROM table is also accessible in Secure memory.

Arm strongly recommends that bits ROMADDR[(PAsize-1):32] are zero in any system that supports AArch32 at the highest implemented Exception level.

ROMADDR encoding when FEAT_LPA is not implemented and MDRAR_EL1.Valid != 0b00

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | ROMADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [39:36]

Reserved, RES0.

ROMADDR, bits [35:0]

Bits [39:12] of the ROM table physical address.

Bits [11:0] of the ROM table physical address are zero.

For implementations with fewer than 48 physical address bits, the corresponding upper bits of this field are RES0.

In an implementation that includes EL3, ROMADDR is an address in Non-secure memory. It is IMPLEMENTATION DEFINED whether the ROM table is also accessible in Secure memory.

Arm strongly recommends that bits ROMADDR[(PAsize-1):32] are zero in any system that supports AArch32 at the highest implemented Exception level.

ROMADDR encoding when MDRAR_EL1.Valid == 0b00

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [39:0]

Reserved, UNKNOWN.

Bits [11:2]

Reserved, RES0.

Valid, bits [1:0]

This field indicates whether the ROM Table address is valid.

| Valid | Meaning |
|-------|---|
| 0b00 | ROM Table address is not valid. Software must ignore ROMADDR. |
| 0b11 | ROM Table address is valid. |

Other values are reserved.

Arm recommends implementations set this field to zero.

Accessing MDRAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MDRAR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDRA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MDRAR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MDRAR_EL1;
elsif PSTATE.EL == EL3 then
    return MDRAR_EL1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MDSCR_EL1, Monitor Debug System Control Register

The MDSCR_EL1 characteristics are:

Purpose

Main control register for the debug implementation.

Configuration

AArch64 System register MDSCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGDSCRext\[31:0\]](#).

AArch64 System register MDSCR_EL1 bit [15] is architecturally mapped to AArch32 System register [DBGDSCRint\[15\]](#).

AArch64 System register MDSCR_EL1 bit [12] is architecturally mapped to AArch32 System register [DBGDSCRint\[12\]](#).

AArch64 System register MDSCR_EL1 bits [5:2] are architecturally mapped to AArch32 System register [DBGDSCRint\[5:2\]](#).

Attributes

MDSCR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|--------|--------|------|-------|-------|--------|-------|------|-----|--------|-----|-----|-----|------|------|-----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TFO | RXfull | TXfull | RES0 | RXOTX | URES0 | INTdis | TDARE | RES0 | SC2 | RAZ/WI | MDE | HDE | KDE | TDCC | RES0 | ERR | RES0 | SS | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

TFO, bit [31]

When FEAT_TRF is implemented:

Trace Filter override. Used for save/restore of [EDSCR.TFO](#).

When [OSLSR_EL1.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [OSLSR_EL1.OSLK](#) == 1, this bit holds the value of [EDSCR.TFO](#). Reads and writes of this bit are indirect accesses to [EDSCR.TFO](#).

Accessing this field has the following behavior:

- When [OSLSR_EL1.OSLK](#) == 1, access to this field is **RW**.
- When [OSLSR_EL1.OSLK](#) == 0, access to this field is **RO**.

Otherwise:

Reserved, RES0.

RXfull, bit [30]

Used for save/restore of [EDSCR.RXfull](#).

When [OSLSR_EL1.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [OSLSR_EL1.OSLK](#) == 1, this bit holds the value of [EDSCR.RXfull](#). Reads and writes of this bit are indirect accesses to [EDSCR.RXfull](#).

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When [OSLSR_EL1.OSLK](#) == 1, access to this field is **RW**.
- When [OSLSR_EL1.OSLK](#) == 0, access to this field is **RO**.

TXfull, bit [29]

Used for save/restore of [EDSCR.TXfull](#).

When [OSLSR_EL1.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [OSLSR_EL1.OSLK](#) == 1, this bit holds the value of [EDSCR.TXfull](#). Reads and writes of this bit are indirect accesses to [EDSCR.TXfull](#).

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When [OSLSR_EL1.OSLK](#) == 1, access to this field is **RW**.
- When [OSLSR_EL1.OSLK](#) == 0, access to this field is **RO**.

Bit [28]

Reserved, RES0.

RXO, bit [27]

Used for save/restore of [EDSCR.RXO](#).

When [OSLSR_EL1.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [OSLSR_EL1.OSLK](#) == 1, this bit holds the value of [EDSCR.RXO](#). Reads and writes of this bit are indirect accesses to [EDSCR.RXO](#).

When [OSLSR_EL1.OSLK](#) == 1, if bits [27,6] of the value written to MDSCR_EL1 are {1,0}, that is, the RXO bit is 1 and the ERR bit is 0, the PE sets [EDSCR.{RXO,ERR}](#) to UNKNOWN values.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When [OSLSR_EL1.OSLK](#) == 1, access to this field is **RW**.
- When [OSLSR_EL1.OSLK](#) == 0, access to this field is **RO**.

TXU, bit [26]

Used for save/restore of [EDSCR.TXU](#).

When [OSLSR_EL1.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [OSLSR_EL1.OSLK](#) == 1, this bit holds the value of [EDSCR.TXU](#). Reads and writes of this bit are indirect accesses to [EDSCR.TXU](#).

When [OSLSR_EL1.OSLK](#) == 1, if bits [26,6] of the value written to MDSCR_EL1 are {1,0}, that is, the TXU bit is 1 and the ERR bit is 0, the PE sets [EDSCR.{TXU,ERR}](#) to UNKNOWN values.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

Bits [25:24]

Reserved, RES0.

INTdis, bits [23:22]

Used for save/restore of [EDSCR](#).INTdis.

When [OSLSR_EL1](#).OSLK == 0, and software must treat this bit as UNK/SBZP.

When [OSLSR_EL1](#).OSLK == 1, this field holds the value of [EDSCR](#).INTdis. Reads and writes of this field are indirect accesses to [EDSCR](#).INTdis.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

TDA, bit [21]

Used for save/restore of [EDSCR](#).TDA.

When [OSLSR_EL1](#).OSLK == 0, software must treat this bit as UNK/SBZP.

When [OSLSR_EL1](#).OSLK == 1, this bit holds the value of [EDSCR](#).TDA. Reads and writes of this bit are indirect accesses to [EDSCR](#).TDA.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

Bit [20]

Reserved, RES0.

SC2, bit [19]

When FEAT_PCSRv8 is implemented, FEAT_VHE is implemented and FEAT_PCSRv8p2 is not implemented:

Used for save/restore of [EDSCR](#).SC2.

When [OSLSR_EL1](#).OSLK == 0, software must treat this bit as UNK/SBZP.

When [OSLSR_EL1](#).OSLK == 1, this bit holds the value of [EDSCR](#).SC2. Reads and writes of this bit are indirect accesses to [EDSCR](#).SC2.

Accessing this field has the following behavior:

- When OSLSR_EL1.OSLK == 1, access to this field is **RW**.
- When OSLSR_EL1.OSLK == 0, access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [18:16]

Reserved, RAZ/WI.

Hardware must implement this field as RAZ/WI. Software must not rely on the register reading as zero, and must use a read-modify-write sequence to write to the register.

MDE, bit [15]

Monitor debug events. Enable Breakpoint, Watchpoint, and Vector Catch exceptions.

| MDE | Meaning |
|-----|---|
| 0b0 | Breakpoint, Watchpoint, and Vector Catch exceptions disabled. |
| 0b1 | Breakpoint, Watchpoint, and Vector Catch exceptions enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HDE, bit [14]

Used for save/restore of [EDSCR.HDE](#).

When [OSLSR_EL1.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [OSLSR_EL1.OSLK](#) == 1, this bit holds the value of [EDSCR.HDE](#). Reads and writes of this bit are indirect accesses to [EDSCR.HDE](#).

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When [OSLSR_EL1.OSLK](#) == 1, access to this field is **RW**.
- When [OSLSR_EL1.OSLK](#) == 0, access to this field is **RO**.

KDE, bit [13]

Local (kernel) debug enable. If EL_D is using AArch64, enable debug exceptions within EL_D . Permitted values are:

| KDE | Meaning |
|-----|--|
| 0b0 | Debug exceptions, other than Breakpoint Instruction exceptions, disabled within EL_D . |
| 0b1 | All debug exceptions enabled within EL_D . |

RES0 if EL_D is using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TDCC, bit [12]

Traps EL_0 accesses to the Debug Communication Channel (DCC) registers to EL_1 , or to EL_2 when it is implemented and enabled for the current Security state and [HCR_EL2.TGE](#) is 1, from both Execution states, as follows:

- In AArch64 state, MRS or MSR accesses to the following DCC registers are trapped, reported using EC syndrome value 0x18:
 - [MDCCSR_EL0](#).
 - If not in Debug state, [DBGDTR_EL0](#), [DBGDTRTX_EL0](#), and [DBGDTRRX_EL0](#).

- In AArch32 state, MRC or MCR accesses to the following registers are trapped, reported using EC syndrome value 0x05.
 - [DBGDSCRint](#), [DBGDIDR](#), [DBGDSAR](#), [DBGDRAR](#).
 - If not in Debug state, [DBGDTRRXint](#), and [DBGDTRTXint](#).
- In AArch32 state, LDC access to [DBGDTRRXint](#) and STC access to [DBGDTRTXint](#) are trapped, reported using EC syndrome value 0x06.
- In AArch32 state, MRRC accesses to [DBGDSAR](#) and [DBGDRAR](#) are trapped, reported using EC syndrome value 0x0C.

| TDCC | Meaning |
|-------------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL0 using AArch64: EL0 accesses to the AArch64 DCC registers are trapped. EL0 using AArch32: EL0 accesses to the AArch32 DCC registers are trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:7]

Reserved, RES0.

ERR, bit [6]

Used for save/restore of [EDSCR](#).ERR.

When [OSLSR_EL1](#).OSLK == 0, software must treat this bit as UNK/SBZP.

When [OSLSR_EL1](#).OSLK == 1, this bit holds the value of [EDSCR](#).ERR. Reads and writes of this bit are indirect accesses to [EDSCR](#).ERR.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When [OSLSR_EL1](#).OSLK == 1, access to this field is **RW**.
- When [OSLSR_EL1](#).OSLK == 0, access to this field is **RO**.

Bits [5:1]

Reserved, RES0.

SS, bit [0]

Software step control bit. If EL_D is using AArch64, enable Software step. Permitted values are:

| SS | Meaning |
|-----------|------------------------|
| 0b0 | Software step disabled |
| 0b1 | Software step enabled. |

RES0 if EL_D is using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MDSCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MDSCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.MDSCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x158];
    else
        return MDSCR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MDSCR_EL1;
elsif PSTATE.EL == EL3 then
    return MDSCR_EL1;

```

MSR MDSCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.MDSCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        NVMem[0x158] = X[t];
    else
        MDSCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MDSCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    MDSCR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MFAR_EL3, PA Fault Address Register

The MFAR_EL3 characteristics are:

Purpose

Holds the faulting physical address for Granule Protection Check exceptions taken to EL3.

Configuration

This register is present only when FEAT_RME is implemented. Otherwise, direct accesses to MFAR_EL3 are UNDEFINED.

Attributes

MFAR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|-----|------|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| NS | NSE | RES0 | | | | | | | | | | FPA[51:48] | | | | | FPA[47:12] | | | | | | | | | | | | | | |
| FPA[47:12] | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

An exception return at EL3 makes MFAR_EL3 UNKNOWN.

NS, bit [63]

Together with the NSE field, reports the physical address space of the access that triggered the Granule Protection Check exception.

| NSE | NS | Meaning |
|-----|-----|-------------|
| 0b0 | 0b0 | Secure. |
| 0b0 | 0b1 | Non-secure. |
| 0b1 | 0b0 | Root. |
| 0b1 | 0b1 | Realm. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSE, bit [62]

Together with the NS field, reports the physical address space of the access that triggered the Granule Protection Check exception.

For a description of the values derived by evaluating NS and NSE together, see MFAR_EL3.NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [61:52]

Reserved, RES0.

FPA[51:48], bits [51:48]**When FEAT_LPA is implemented:**

When FEAT_LPA is implemented, extension to FPA[47:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FPA[47:12], bits [47:12]

Bits [47:12] of the faulting physical address.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

Accessing MFAR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MFAR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0110 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return MFAR_EL3;

```

MSR MFAR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0110 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    MFAR_EL3 = X[t];

```


MIDR_EL1, Main ID Register

The MIDR_EL1 characteristics are:

Purpose

Provides identification information for the PE, including an implementer code for the device and a device ID number.

Configuration

AArch64 System register MIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [MIDR\[31:0\]](#).

AArch64 System register MIDR_EL1 bits [31:0] are architecturally mapped to External register [MIDR_EL1\[31:0\]](#).

Attributes

MIDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|---------|----|----|----|--------------|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Implementer | | | | | | | | Variant | | | | Architecture | | | | PartNum | | | | | | | | | | | | | | Revision | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:32]

Reserved, RES0.

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

| Implementer | Meaning |
|-------------|--|
| 0x00 | Reserved for software use. |
| 0x41 | Arm Limited. |
| 0x42 | Broadcom Corporation. |
| 0x43 | Cavium Inc. |
| 0x44 | Digital Equipment Corporation. |
| 0x46 | Fujitsu Ltd. |
| 0x49 | Infineon Technologies AG. |
| 0x4D | Motorola or Freescale Semiconductor Inc. |
| 0x4E | NVIDIA Corporation. |
| 0x50 | Applied Micro Circuits Corporation. |
| 0x51 | Qualcomm Inc. |
| 0x56 | Marvell International Ltd. |
| 0x69 | Intel Corporation. |
| 0xC0 | Ampere Computing. |

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

Access to this field is **RO**.

Variant, bits [23:20]

Variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Architecture, bits [19:16]

Architecture version. Defined values are:

| Architecture | Meaning |
|--------------|---|
| 0b0001 | Armv4. |
| 0b0010 | Armv4T. |
| 0b0011 | Armv5 (obsolete). |
| 0b0100 | Armv5T. |
| 0b0101 | Armv5TE. |
| 0b0110 | Armv5TEJ. |
| 0b0111 | Armv6. |
| 0b1111 | Architectural features are individually identified in the ID_* registers. |

All other values are reserved.

Access to this field is **RO**.

PartNum, bits [15:4]

Primary Part Number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [3:0]

Revision number for the device.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.MIDR_EL1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() then
            return VPIDR_EL2;
        else
            return MIDR_EL1;
    elsif PSTATE.EL == EL2 then
        return MIDR_EL1;
    elsif PSTATE.EL == EL3 then
        return MIDR_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAM0_EL1, MPAM0 Register (EL1)

The MPAM0_EL1 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL0. When EL2 is implemented and enabled in the current Security state, the MPAM virtualization option is present, [MPAMHCR_EL2.GSTAPP_PLK](#) == 1 and [HCR_EL2.TGE](#) == 0, [MPAM1_EL1](#) is used instead of MPAM0_EL1 to generate MPAM information to label memory requests.

If EL2 is implemented and enabled in the current Security state, and [HCR_EL2.E2H](#) == 0 or [HCR_EL2.TGE](#) == 0, the MPAM virtualization option is present and [MPAMHCR_EL2.EL0_VPMEN](#) == 1, then MPAM PARTIDs in MPAM0_EL1 are virtual and mapped into physical PARTIDs for the current Security state.

Configuration

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM0_EL1 are UNDEFINED.

Attributes

MPAM0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | PMG_D | | | | | | | | PMG_I | | | | | | | |
| PARTID_D | | | | | | | | | | | | | | | | PARTID_I | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group property for PARTID_I.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAM0_EL1

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAM0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM0EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return MPAM0_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAM0_EL1;
elsif PSTATE.EL == EL3 then
    return MPAM0_EL1;

```

MSR MPAM0_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM0EL1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            MPAM0_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                MPAM0_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        MPAM0_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAM1_EL1, MPAM1 Register (EL1)

The MPAM1_EL1 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL1.

When EL2 is implemented and enabled in the current Security state, the MPAM virtualization option is present, [MPAMHCR_EL2.GSTAPP_PLK](#) == 1 and [HCR_EL2.TGE](#) == 0, MPAM1_EL1 is used instead of [MPAM0_EL1](#) to generate MPAM labels for memory requests when executing at EL0.

MPAM1_EL1 is an alias for [MPAM2_EL2](#) when executing at EL2 with [HCR_EL2.E2H](#) == 1.

MPAM1_EL12 is an alias for MPAM1_EL1 when executing at EL2 or EL3 with [HCR_EL2.E2H](#) == 1.

If EL2 is implemented and enabled in the current Security state, the MPAM virtualization option is present and [MPAMHCR_EL2.EL1_VPMEN](#) == 1, MPAM PARTIDs in MPAM1_EL1 are virtual and mapped into physical PARTIDs for the current Security state. This mapping of MPAM1_EL1 virtual PARTIDs to physical PARTIDs when EL1_VPMEN is 1 also applies when MPAM1_EL1 is used at EL0 due to [MPAMHCR_EL2.GSTAPP_PLK](#).

Configuration

AArch64 System register MPAM1_EL1 bit [63] is architecturally mapped to AArch64 System register [MPAM3_EL3\[63\]](#) when EL3 is implemented.

AArch64 System register MPAM1_EL1 bit [63] is architecturally mapped to AArch64 System register [MPAM2_EL2\[63\]](#) when EL3 is not implemented and EL2 is implemented.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM1_EL1 are UNDEFINED.

Attributes

MPAM1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|------|-----------|------|------------|------|-------|-------|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| MPAMEN | RES0 | FORCED_NS | RES0 | ALTSP_FRCD | RES0 | PMG_D | PMG_I | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PARTID_D | | | | | | | | | | | | | | | | PARTID_I | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

MPAMEN, bit [63]

MPAM Enable. MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

| MPAMEN | Meaning |
|--------|--|
| 0b0 | The default PARTID and default PMG are output in MPAM information. |
| 0b1 | MPAM information is output based on the MPAMn_ELx register for ELn according the MPAM configuration. |

If neither EL3 nor EL2 is implemented, this field is read/write.

If EL3 is implemented, this field is read-only and reads the current value of the read/write bit [MPAM3_EL3.MPAMEN](#).

If EL3 is not implemented and EL2 is implemented, this field is read-only and reads the current value of the read/write bit [MPAM2_EL2.MPAMEN](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing this field has the following behavior:

- **RW** if all of the following are true:
 - EL3 is not implemented
 - EL2 is not implemented
- Otherwise, access to this field is **RO**.

Bits [62:61]

Reserved, RES0.

FORCED_NS, bit [60]

When FEAT_MPAMv0p1 is implemented:

In the Secure state, FORCED_NS indicates the state of [MPAM3_EL3.FORCE_NS](#).

| FORCED_NS | Meaning |
|-----------|---|
| 0b0 | In the Non-secure state, always reads as 0. In the Secure state, indicates that MPAM3_EL3.FORCE_NS == 0. |
| 0b1 | In the Secure state, indicates that MPAM3_EL3.FORCE_NS == 1. |

Always reads as 0 in the Non-secure state.

Writes are ignored.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [59:55]

Reserved, RES0.

ALTSP_FRCD, bit [54]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Alternative PARTID forced for PARTIDs in this register.

| ALTSP_FRCD | Meaning |
|------------|--|
| 0b0 | The PARTIDs in MPAM1_EL1 and MPAM0_EL1 are using the primary PARTID space. |
| 0b1 | The PARTIDs in MPAM1_EL1 and MPAM0_EL1 are using the alternative PARTID space. |

This bit indicates that a higher Exception level has forced the PARTIDs in this register to use the alternative PARTID space defined for the current Security state.

In MPAM1_EL1, it also indicates that [MPAM0_EL1](#) is forced to use alternative PARTID space.

For more information, see 'Alternative PARTID spaces and selection' in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598).

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [53:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group property for PARTID_I.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAM1_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, accesses from EL3 using the mnemonic MPAM1_EL1 or MPAM1_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAM1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            return NVMem[0x900];
        else
            return MPAM1_EL1;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            elsif HCR_EL2.E2H == '1' then
                return MPAM2_EL2;
            else
                return MPAM1_EL1;
    elsif PSTATE.EL == EL3 then
        return MPAM1_EL1;

```

MSR MPAM1_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            NVMem[0x900] = X[t];
        else
            MPAM1_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            elsif HCR_EL2.E2H == '1' then
                MPAM2_EL2 = X[t];
            else
                MPAM1_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        MPAM1_EL1 = X[t];

```

MRS <Xt>, MPAM1_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1010 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x900];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' then
            if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return MPAM1_EL1;
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
            return MPAM1_EL1;
        else
            UNDEFINED;

```

MSR MPAM1_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1010 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x900] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.E2H == '1' then
            if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            else
                MPAM1_EL1 = X[t];
        else
            UNDEFINED;
    elsif PSTATE.EL == EL3 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
            MPAM1_EL1 = X[t];
        else
            UNDEFINED;

```


MPAM2_EL2, MPAM2 Register (EL2)

The MPAM2_EL2 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL2.

Configuration

AArch64 System register MPAM2_EL2 bit [63] is architecturally mapped to AArch64 System register [MPAM3_EL3\[63\]](#) when EL3 is implemented.

AArch64 System register MPAM2_EL2 bit [63] is architecturally mapped to AArch64 System register [MPAM1_EL1\[63\]](#).

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM2_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAM2_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | |
|----------|----------|------|------|-----------|-----------|------------|--------|----------|--------------|--------------|-----|
| 63 | 62616059 | 58 | 57 | 56 | 55 | 54 | 535251 | 50 | 49 | 48 | 474 |
| MPAMEN | RES0 | TIDR | RES0 | ALTSP_HFC | ALTSP_EL2 | ALTSP_FRCD | RES0 | EnMPAMSM | TRAPMPAM0EL1 | TRAPMPAM1EL1 | |
| PARTID_D | | | | | | | | | | | |
| 31 | 30292827 | 26 | 25 | 24 | 23 | 22 | 212019 | 18 | 17 | 16 | 151 |

MPAMEN, bit [63]

MPAM Enable. MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

| MPAMEN | Meaning |
|--------|---|
| 0b0 | The default PARTID and default PMG are output in MPAM information from all Exception levels. |
| 0b1 | MPAM information is output based on the MPAMn_ELx register for ELn according to the MPAM configuration. |

If EL3 is not implemented, this field is read/write.

If EL3 is implemented, this field is read-only and reads the current value of the read/write [MPAM3_EL3.MPAMEN](#) bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing this field has the following behavior:

- When EL3 is not implemented, access to this field is **RW**.
- Otherwise, access to this field is **RO**.

Bits [62:59]

Reserved, RES0.

TIDR, bit [58]

When (FEAT_MPAMvOp1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMIDR_EL1.HAS_TIDR == 1:

TIDR traps accesses to [MPAMIDR_EL1](#) from EL1 to EL2.

| TIDR | Meaning |
|------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Trap accesses to MPAMIDR_EL1 from EL1 to EL2. |

[MPAMHCR_EL2](#).TRAP_MPAMIDR_EL1 == 1 also traps [MPAMIDR_EL1](#) accesses from EL1 to EL2. If either TIDR or TRAP_MPAMIDR_EL1 are 1, accesses are trapped.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [57]

Reserved, RES0.

ALTSP_HFC, bit [56]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Hierarchical force of alternative PARTID space controls. When [MPAM3_EL3](#).ALTSP_HEN is 0, ALTSP controls in MPAM2_EL2 have no effect. When MPAM3_EL3.ALTSP_HEN is 1, this bit selects whether the PARTIDs in [MPAM1_EL1](#) and [MPAM0_EL1](#) are in the primary (0) or alternative (1) PARTID space for the security state.

| ALTSP_HFC | Meaning |
|-----------|--|
| 0b0 | When MPAM3_EL3 .ALTSP_HEN is 1, the PARTID space of MPAM1_EL1 .PARTID_I, MPAM1_EL1 .PARTID_D, MPAM0_EL1 .PARTID_I, and MPAM0_EL1 .PARTID_D are in the primary PARTID space for the Security state. |
| 0b1 | When MPAM3_EL3 .ALTSP_HEN is 1, the PARTID space of MPAM1_EL1 .PARTID_I, MPAM1_EL1 .PARTID_D, MPAM0_EL1 .PARTID_I, and MPAM0_EL1 .PARTID_D are in the alternative PARTID space for the Security state. |

This control has no effect when [MPAM3_EL3](#).ALTSP_HEN is 0.

For more information, see 'Alternative PARTID spaces and selection' in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_EL2, bit [55]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Select alternative PARTID space for PARTIDs in MPAM2_EL2 when [MPAM3_EL3](#).ALTSP_HEN is 1.

| ALTSP_EL2 | Meaning |
|------------------|--|
| 0b0 | When MPAM3_EL3.ALTSP_HEN is 1, selects the primary PARTID space for MPAM2_EL2.PARTID_I and MPAM2_EL2.PARTID_D. |
| 0b1 | When MPAM3_EL3.ALTSP_HEN is 1, selects the alternative PARTID space for MPAM2_EL2.PARTID_I and MPAM2_EL2.PARTID_D. |

For more information, see 'Alternative PARTID spaces and selection' in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_FRCD, bit [54]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Alternative PARTID forced for PARTIDs in this register.

| ALTSP_FRCD | Meaning |
|-------------------|--|
| 0b0 | The PARTIDs in this register are using the primary PARTID space. |
| 0b1 | The PARTIDs in this register are using the alternative PARTID space. |

This bit indicates that a higher Exception level has forced the PARTIDs in this register to use the alternative PARTID space defined for the current Security state. In EL2, it is also 1 when MPAM2_EL2.ALTSP_EL2 is 1.

For more information, see 'Alternative PARTID spaces and selection' in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [53:51]

Reserved, RES0.

EnMPAMSM, bit [50]

When FEAT_SME is implemented:

Traps execution at EL1 of instructions that directly access the [MPAMSM_EL1](#) register to EL2. The exception is reported using ESR_ELx.EC value 0x18.

| EnMPAMSM | Meaning |
|-----------------|---|
| 0b0 | This control causes execution of these instructions at EL1 to be trapped. |
| 0b1 | This control does not cause execution of any instructions to be trapped. |

This field has no effect on accesses to [MPAMSM_EL1](#) from EL2 or EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRAPMPAM0EL1, bit [49]

Trap accesses from EL1 to the [MPAM0_EL1](#) register trap to EL2.

| TRAPMPAM0EL1 | Meaning |
|--------------|--|
| 0b0 | Accesses to MPAM0_EL1 from EL1 are not trapped. |
| 0b1 | Accesses to MPAM0_EL1 from EL1 are trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset:
 - When EL3 is not implemented, this field resets to 1.
 - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

TRAPMPAM1EL1, bit [48]

Trap accesses from EL1 to the [MPAM1_EL1](#) register trap to EL2.

| TRAPMPAM1EL1 | Meaning |
|--------------|--|
| 0b0 | Accesses to MPAM1_EL1 from EL1 are not trapped. |
| 0b1 | Accesses to MPAM1_EL1 from EL1 are trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset:
 - When EL3 is not implemented, this field resets to 1.
 - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

PMG_D, bits [47:40]

Performance monitoring group for data accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group for instruction accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAM2_EL2

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAM2_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAM2_EL2;
elsif PSTATE.EL == EL3 then
    return MPAM2_EL2;

```

MSR MPAM2_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            end
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        end
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        end
    else
        MPAM2_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    MPAM2_EL2 = X[t];

```

MRS <Xt>, MPAM1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        end
    elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x900];
    else
        return MPAM1_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        end
    elsif HCR_EL2.E2H == '1' then
        return MPAM2_EL2;
    else
        return MPAM1_EL1;
elsif PSTATE.EL == EL3 then
    return MPAM1_EL1;

```

MSR MPAM1_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.TRAPMPAM1EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x900] = X[t];
    else
        MPAM1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        MPAM2_EL2 = X[t];
    else
        MPAM1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    MPAM1_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAM3_EL3, MPAM3 Register (EL3)

The MPAM3_EL3 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests when executing at EL3.

Configuration

AArch64 System register MPAM3_EL3 bit [63] is architecturally mapped to AArch64 System register [MPAM2_EL2\[63\]](#) when EL2 is implemented.

AArch64 System register MPAM3_EL3 bit [63] is architecturally mapped to AArch64 System register [MPAM1_EL1\[63\]](#).

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAM3_EL3 are UNDEFINED.

Attributes

MPAM3_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------|----|------------------|----|---------------|----|-----------------|----|-------------|----|------------------|----|------------------|----|------------------|----|-------------|----|--------------------|----|-------------|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| <u>MPAMEN</u> | | <u>TRAPLOWER</u> | | <u>SDEFLT</u> | | <u>FORCE_NS</u> | | <u>RESO</u> | | <u>ALTSP_HEN</u> | | <u>ALTSP_HFC</u> | | <u>ALTSP_EL3</u> | | <u>RESO</u> | | <u>RT_ALTSP_NS</u> | | <u>RESO</u> | | | | | | | | | | | | <u>PMG_</u> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| <u>PARTID_D</u> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

MPAMEN, bit [63]

MPAM Enable. MPAM is enabled when MPAMEN == 1. When disabled, all PARTIDs and PMGs are output as their default value in the corresponding ID space.

Values of this field are:

| MPAMEN | Meaning |
|---------------|--|
| 0b0 | The default PARTID and default PMG are output in MPAM information when executing at any ELn. |
| 0b1 | MPAM information is output based on the MPAMn_ELx register for ELn according the MPAM configuration. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Access to this field is **RW**.

TRAPLOWER, bit [62]

Trap direct accesses to MPAM System registers that are not UNDEFINED from all ELn lower than EL3.

| TRAPLOWER | Meaning |
|-----------|---|
| 0b0 | Do not force trapping of direct accesses of MPAM System registers to EL3. |
| 0b1 | Force direct accesses of MPAM System registers to trap to EL3. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

SDEFLT, bit [61]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMIDR_EL1.HAS_SDEFLT == 1:

SDEFLT overrides the PARTID and PMG with the default PARTID and default PMG when executing in the Secure state.

| SDEFLT | Meaning |
|--------|---|
| 0b0 | The PARTID and PMG are determined normally in the Secure state. |
| 0b1 | When executing in the Secure state, the PARTID is always PARTID 0, and the PMG is always PMG 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FORCE_NS, bit [60]

When FEAT_MPAMv0p1 is implemented and MPAMIDR_EL1.HAS_FORCE_NS == 1:

FORCE_NS forces MPAM_NS to always be 1 in the Secure state.

| FORCE_NS | Meaning |
|----------|--|
| 0b0 | MPAM_NS is 0 when executing in the Secure state. |
| 0b1 | MPAM_NS is 1 when executing in the Secure state. |

An implementation is permitted to have this field as RAO if the implementation does not support generating MPAM_NS as 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [59:58]

Reserved, RES0.

ALTSP_HEN, bit [57]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Hierarchical enable for alternative PARTID space controls. Alternative PARTID space controls in [MPAM2_EL2](#) have no effect when this field is zero.

| ALTSP_HEN | Meaning |
|-----------|--|
| 0b0 | Disable alternative PARTID space controls in MPAM2_EL2 . The PARTID space for PARTIDs in MPAM2_EL2 , MPAM1_EL1 , and MPAM0_EL1 is selected by MPAM3_EL3.ALTSP_HFC. |
| 0b1 | Enable alternative PARTID space controls in MPAM2_EL2 to control the PARTID space used for PARTIDs in MPAM2_EL2 , MPAM1_EL1 , and MPAM0_EL1 . |

For more information, see 'Alternative PARTID spaces and selection' in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_HFC, bit [56]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Hierarchical force of alternative PARTID space controls. When MPAM3_EL3.ALTSP_HEN is 0, the PARTID space for PARTIDs in [MPAM2_EL2](#), [MPAM1_EL1](#), and [MPAM0_EL1](#) is selected by the value of this bit.

| ALTSP_HFC | Meaning |
|-----------|---|
| 0b0 | When MPAM3_EL3.ALTSP_HEN is 0, the PARTID space of MPAM2_EL2 .PARTID, MPAM1_EL1 .PARTID and MPAM0_EL1 .PARTID are the primary PARTID space for the security state. |
| 0b1 | When MPAM3_EL3.ALTSP_HEN is 0, the PARTID space of MPAM2_EL2 .PARTID and MPAM1_EL1 .PARTID and MPAM0_EL1 .PARTID are the alternative PARTID space for the security state. |

For more information, see 'Alternative PARTID spaces and selection' in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ALTSP_EL3, bit [55]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Select alternative PARTID space for PARTIDs in MPAM3_EL3.

| ALTSP_EL3 | Meaning |
|-----------|---|
| 0b0 | Selects the primary PARTID space of MPAM3_EL3 .PARTID_I and MPAM3_EL3.PARTID_D. |
| 0b1 | Selects the alternative PARTID space of MPAM3_EL3 .PARTID_I and MPAM3_EL3.PARTID_D. |

For more information, see 'Alternative PARTID spaces and selection' in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [54:53]

Reserved, RES0.

RT_ALTSP_NS, bit [52]

When FEAT_RME is implemented and MPAMIDR_EL1.HAS_ALTSP == 1:

Alternative PARTID space selection for the Root security state.

| RT_ALTSP_NS | Meaning |
|-------------|---|
| 0b0 | The alternative PARTID space in the Root security state is the Secure PARTID space. |
| 0b1 | The alternative PARTID space in the Root security state is the Non-secure PARTID space. |

This field has no effect except in the Root security state (EL3).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [51:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group for data accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PMG_I, bits [39:32]

Performance monitoring group for instruction accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_D, bits [31:16]

Partition ID for data accesses, including load and store accesses, made from EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PARTID_I, bits [15:0]

Partition ID for instruction accesses made from EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAM3_EL3

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAM3_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1010 | 0b0101 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return MPAM3_EL3;
```

MSR MPAM3_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1010 | 0b0101 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    MPAM3_EL3 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMHCR_EL2, MPAM Hypervisor Control Register (EL2)

The MPAMHCR_EL2 characteristics are:

Purpose

Controls the PARTID virtualization features of MPAM. It controls the mapping of virtual PARTIDs into physical PARTIDs in [MPAM0_EL1](#) when EL0_VPMEN == 1 and in [MPAM1_EL1](#) when EL1_VPMEN == 1.

Configuration

This register is present only when FEAT_MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMHCR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMHCR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TRAP_MPAMIDR_EL1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

TRAP_MPAMIDR_EL1, bit [31]

Trap accesses from EL1 to [MPAMIDR_EL1](#) to EL2.

| TRAP_MPAMIDR_EL1 | Meaning |
|------------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Direct accesses to MPAMIDR_EL1 from EL1 are trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset:
 - When EL3 is not implemented, this field resets to 1.
 - When EL3 is implemented, this field resets to an architecturally UNKNOWN value.

Bits [30:9]

Reserved, RES0.

GSTAPP_PLK, bit [8]

Make the PARTIDs at EL0 the same as the PARTIDs at EL1. When executing at EL0, EL2 is enabled, [HCR_EL2.TGE](#) == 0 and [GSTAPP_PLK](#) = 1, [MPAM1_EL1](#) is used instead of [MPAM0_EL1](#) to generate MPAM labels for memory requests.

| GSTAPP_PLK | Meaning |
|-------------------|---|
| 0b0 | MPAM0_EL1 is used to generate MPAM labels when executing at EL0. |
| 0b1 | MPAM1_EL1 is used to generate MPAM labels when executing at EL0 with EL2 enabled and HCR_EL2.TGE == 0. Otherwise MPAM0_EL1 is used. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:2]

Reserved, RES0.

EL1_VPMEN, bit [1]

Enable the virtual PARTID mapping of the PARTID fields in [MPAM1_EL1](#) when executing at EL1. This bit also enables virtual PARTID mapping when [MPAM1_EL1](#) is used to generate MPAM labels for memory requests at EL0 due to [GSTAPP_PLK](#) == 1.

| EL1_VPMEN | Meaning |
|------------------|---|
| 0b0 | MPAM1_EL1 .PARTID_I and MPAM1_EL1 .PARTID_D are physical PARTIDs that are used to label memory system requests. |
| 0b1 | MPAM1_EL1 .PARTID_I and MPAM1_EL1 .PARTID_D are virtual PARTIDs that are used to index the PhyPARTID fields of MPAMVPM0_EL2 to MPAMVPM7_EL2 registers to map the virtual PARTID into a physical PARTID to label memory system requests. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EL0_VPMEN, bit [0]

Enable the virtual PARTID mapping of the PARTID fields of [MPAM0_EL1](#) unless [HCR_EL2.E2H](#) == 1 and [HCR_EL2.TGE](#) == 1.

When [HCR_EL2.E2H](#) == 1 and [HCR_EL2.TGE](#) == 1, [EL0_VPMEN](#) is ignored and [MPAM0_EL1](#) PARTID fields are not mapped.

When [MPAMHCR_EL2.GSTAPP_PLK](#) == 1 and [HCR_EL2.TGE](#) == 0, [MPAM1_EL1](#) is used as the source of PARTIDs and the virtual PARTID mapping of [MPAM1_EL1](#) PARTIDs is controlled by [MPAMHCR_EL2.EL1_VPMEN](#).

| EL0_VPMEN | Meaning |
|------------------|---|
| 0b0 | MPAM0_EL1 .PARTID_I and MPAM0_EL1 .PARTID_D are physical PARTIDs that are used to label memory system requests. |
| 0b1 | MPAM0_EL1 .PARTID_I and MPAM0_EL1 .PARTID_D are virtual PARTIDs that are used to index the PhyPARTID fields of MPAMVPM0_EL2 to MPAMVPM7_EL2 registers to map the virtual PARTID into a physical PARTID to label memory system requests. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMHCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAMHCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x930];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return MPAMHCR_EL2;
    elsif PSTATE.EL == EL3 then
        return MPAMHCR_EL2;

```

MSR MPAMHCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x930] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAMHCR_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        MPAMHCR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMIDR_EL1, MPAM ID Register (EL1)

The MPAMIDR_EL1 characteristics are:

Purpose

Indicates the presence and maximum PARTID and PMG values supported in the implementation. It also indicates whether the implementation supports MPAM virtualization.

Configuration

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMIDR_EL1 are UNDEFINED.

Attributes

MPAMIDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------------|--------------|-----|----------|-----------|------|----|----|----|----------|----|---------|----|------|-----------|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 |
| RES0 | HAS_SDEFLT | HAS_FORCE_NS | SP4 | HAS_TIDR | HAS_ALTSP | RES0 | | | | | | | | | | P | | | | | | | | | |
| RES0 | | | | | | | | | | VPMR_MAX | | HAS_HCR | | RES0 | PARTID_MA | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 |

MPAMIDR_EL1 indicates the MPAM implementation parameters of the PE.

Bits [63:62]

Reserved, RES0.

HAS_SDEFLT, bit [61]

HAS_SDEFLT indicates support for [MPAM3_EL3](#).SDEFLT bit. Defined values are:

| HAS_SDEFLT | Meaning |
|------------|--|
| 0b0 | The SDEFLT bit is not implemented in MPAM3_EL3 . |
| 0b1 | The SDEFLT bit is implemented in MPAM3_EL3 . |

When `MPAM3_EL3.SDEFLT == 1`, accesses from the Secure Execution state use the default PARTID, `PARTID == 0`.

HAS_FORCE_NS, bit [60]

HAS_FORCE_NS indicates support for [MPAM3_EL3.FORCE_NS](#) bit. Defined values are:

| HAS_FORCE_NS | Meaning |
|--------------|--|
| 0b0 | The FORCE_NS bit is not implemented in MPAM3_EL3 . |
| 0b1 | The FORCE_NS bit is implemented in MPAM3_EL3 . |

When [MPAM3_EL3.FORCE](#) NS == 1, accesses from the Secure Execution state have MPAM NS == 1.

SP4, bit [59]

Supports 4 MPAM PARTID spaces.

| SP4 | Meaning |
|-----|--------------------------------|
| 0b0 | MPAM supports 2 PARTID spaces. |
| 0b1 | MPAM supports 4 PARTID spaces. |

HAS_TIDR, bit [58]

HAS_TIDR indicates support for [MPAM2_EL2.TIDR](#) bit. Defined values are:

| HAS_TIDR | Meaning |
|----------|--|
| 0b0 | The TIDR bit is not implemented in MPAM2_EL2 . |
| 0b1 | The TIDR bit is implemented in MPAM2_EL2 . |

Note

Arm recommends that when the MPAM version is MPAM v0.1 or MPAM v1.1, MPAMIDR_EL1.HAS_TIDR is 1 and that the MPAM2_EL2.TIDR field is implemented.

HAS_ALTSP, bit [57]

HAS_ALTSP indicates support for alternative PARTID spaces.

| HAS_ALTSP | Meaning |
|-----------|--|
| 0b0 | Alternative PARTID spaces are not implemented. |
| 0b1 | Alternative PARTID spaces are implemented with control bits in MPAM3_EL3 and MPAM2_EL2 . |

Bits [56:40]

Reserved, RES0.

PMG_MAX, bits [39:32]

The largest value of PMG that the implementation can generate. The PMG_I and PMG_D fields of every MPAMn_ELx must implement at least enough bits to represent PMG_MAX.

Bits [31:21]

Reserved, RES0.

VPMR_MAX, bits [20:18]

When MPAMIDR_EL1.HAS_HCR == 1:

Indicates the maximum register index n for the MPAMVPM<n>_EL2 registers.

Otherwise:

Reserved, RAZ.

HAS_HCR, bit [17]

HAS_HCR indicates that the PE implementation supports MPAM virtualization, including [MPAMHCR_EL2](#), [MPAMVPMV_EL2](#), and MPAMVPM<n>_EL2 with n in the range 0 to VPMR_MAX. Must be 0 if EL2 is not implemented in either Security state.

| HAS_HCR | Meaning |
|---------|---------------------------------------|
| 0b0 | MPAM virtualization is not supported. |
| 0b1 | MPAM virtualization is supported. |

Bit [16]

Reserved, RES0.

PARTID_MAX, bits [15:0]

The largest value of PARTID that the implementation can generate. The PARTID_I and PARTID_D fields of every MPAMn_ELx must implement at least enough bits to represent PARTID_MAX.

Accessing MPAMIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAMIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAMIDR_EL1.HAS_HCR == '1' && MPAMHCR_EL2.TRAP_MPAMIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MPAMIDR_EL1.HAS_TIDR == '1' && MPAM2_EL2.TIDR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return MPAMIDR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMIDR_EL1;
elsif PSTATE.EL == EL3 then
    return MPAMIDR_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMSM_EL1, MPAM Streaming Mode Register

The MPAMSM_EL1 characteristics are:

Purpose

Holds information to generate MPAM labels for memory requests issued by SME load and store instructions and, when the PE is in Streaming SVE mode, by SVE and SIMD&FP load and store instructions. For those requests, the MPAM labels in this register have precedence over the labels in [MPAM0_EL1](#), [MPAM1_EL1](#), [MPAM2_EL2](#), and [MPAM3_EL3](#).

It is IMPLEMENTATION DEFINED whether the MPAM labels in this register are used for memory requests due to hardware page table walks or updates performed as a result of SME load and store instructions and, when the PE is in Streaming SVE mode, SVE and SIMD&FP load and store instructions, and SVE prefetch instructions.

The MPAM labels in this register are only used if [MPAM1_EL1](#).MPAMEN is 1.

For memory requests issued from EL0, the MPAM PARTID in this register is virtual and mapped into a physical PARTID when all of the following are true:

- EL2 is implemented and enabled in the current Security state, and [HCR_EL2](#).{E2H, TGE} != {1, 1}.
- The MPAM virtualization option is implemented and [MPAMHCR_EL2](#).EL0_VPMEN == 1.

For memory requests issued from EL1, the MPAM PARTID in this register is virtual and mapped into a physical PARTID when all of the following are true:

- EL2 is implemented and enabled in the current Security state.
- The MPAM virtualization option is implemented and [MPAMHCR_EL2](#).EL1_VPMEN == 1.

Configuration

This register is present only when FEAT_MPAM is implemented and FEAT_SME is implemented. Otherwise, direct accesses to MPAMSM_EL1 are UNDEFINED.

Attributes

MPAMSM_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| RES0 | | | | | | | | | | | | | | | | PMG_D | | | | | | | | RES0 | | | | | | | | | | | |
| PARTID_D | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

Bits [63:48]

Reserved, RES0.

PMG_D, bits [47:40]

Performance monitoring group property for PARTID_D.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [39:32]

Reserved, RES0.

PARTID_D, bits [31:16]

Partition ID for requests issued due to the execution at any Exception level of SME load and store instructions and, when the PE is in Streaming SVE mode, SVE and SIMD&FP load and store instructions and SVE prefetch instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:0]

Reserved, RES0.

Accessing MPAMSM_EL1

None of the fields in this register are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAMSM_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0101 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.EnMPAMSM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return MPAMSM_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMSM_EL1;
elsif PSTATE.EL == EL3 then
    return MPAMSM_EL1;

```

MSR MPAMSM_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1010 | 0b0101 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && MPAM2_EL2.EnMPAMSM == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        MPAMSM_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        MPAMSM_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    MPAMSM_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM0_EL2, MPAM Virtual PARTID Mapping Register 0

The MPAMVPM0_EL2 characteristics are:

Purpose

MPAMVPM0_EL2 provides mappings from virtual PARTIDs 0 - 3 to physical PARTIDs.

[MPAMIDR_EL1.VPMR_MAX](#) field gives the index of the highest implemented MPAMVPM<n>_EL2 register. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If [MPAMIDR_EL1.VPMR_MAX](#) == 0, there is only a single MPAMVPM<n>_EL2 register, [MPAMVPM0_EL2](#).

Virtual PARTID mapping is enabled by [MPAMHCR_EL2.EL1_VPMEN](#) for PARTIDs in [MPAM1_EL1](#) and by [MPAMHCR_EL2.EL0_VPMEN](#) for PARTIDs in [MPAM0_EL1](#).

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the [MPAMVPMV_EL2.VPM_V](#) bit in bit position n is set to 1.

Configuration

This register is present only when FEAT_MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMVPM0_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM0_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| PhyPARTID3 | | | | | | | | | | | | | | | | PhyPARTID2 | | | | | | | | | | | | | | | |
| PhyPARTID1 | | | | | | | | | | | | | | | | PhyPARTID0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PhyPARTID3, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 3. PhyPARTID3 gives the mapping of virtual PARTID 3 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID2, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 2. PhyPARTID2 gives the mapping of virtual PARTID 2 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID1, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 1. PhyPARTID1 gives the mapping of virtual PARTID 1 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID0, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 0. PhyPARTID0 gives the mapping of virtual PARTID 0 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM0_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAMVPM0_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x940];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMVPM0_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPM0_EL2;

```

MSR MPAMVPM0_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x940] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAMVPM0_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        MPAMVPM0_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM1_EL2, MPAM Virtual PARTID Mapping Register 1

The MPAMVPM1_EL2 characteristics are:

Purpose

MPAMVPM1_EL2 provides mappings from virtual PARTIDs 4 - 7 to physical PARTIDs.

[MPAMIDR_EL1](#).VPMR_MAX field gives the index of the highest implemented [MPAMVPM0_EL2](#) to [MPAMVPM7_EL2](#) registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If [MPAMIDR_EL1](#).VPMR_MAX == 0, there is only a single MPAMVPM<n>_EL2 register, [MPAMVPM0_EL2](#).

Virtual PARTID mapping is enabled by [MPAMHCR_EL2](#).EL1_VPMEN for PARTIDs in [MPAM1_EL1](#) and by [MPAMHCR_EL2](#).EL0_VPMEN for PARTIDs in [MPAM0_EL1](#).

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the [MPAMVPMV_EL2](#).VPM_V bit in bit position n is set to 1.

Configuration

This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 0. Otherwise, direct accesses to MPAMVPM1_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM1_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| PhyPARTID7 | | | | | | | | | | | | | | | | PhyPARTID6 | | | | | | | | | | | | | | | |
| PhyPARTID5 | | | | | | | | | | | | | | | | PhyPARTID4 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PhyPARTID7, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 7. PhyPARTID7 gives the mapping of virtual PARTID 7 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID6, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 6. PhyPARTID6 gives the mapping of virtual PARTID 6 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID5, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 5. PhyPARTID5 gives the mapping of virtual PARTID 5 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID4, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 4. PhyPARTID4 gives the mapping of virtual PARTID 4 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM1_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAMVPM1_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x948];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return MPAMVPM1_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPM1_EL2;

```

MSR MPAMVPM1_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x948] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAMVPM1_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        MPAMVPM1_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM2_EL2, MPAM Virtual PARTID Mapping Register 2

The MPAMVPM2_EL2 characteristics are:

Purpose

MPAMVPM2_EL2 provides mappings from virtual PARTIDs 8 - 11 to physical PARTIDs.

[MPAMIDR_EL1.VPMR_MAX](#) field gives the index of the highest implemented [MPAMVPM0_EL2](#) to [MPAMVPM7_EL2](#) registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If [MPAMIDR_EL1.VPMR_MAX](#) == 0, there is only a single MPAMVPM<n>_EL2 register, [MPAMVPM0_EL2](#).

Virtual PARTID mapping is enabled by [MPAMHCR_EL2.EL1_VPMEN](#) for PARTIDs in [MPAM1_EL1](#) and by [MPAMHCR_EL2.EL0_VPMEN](#) for PARTIDs in [MPAM0_EL1](#).

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the [MPAMVPMV_EL2.VPM_V](#) bit in bit position n is set to 1.

Configuration

This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 1. Otherwise, direct accesses to MPAMVPM2_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM2_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| PhyPARTID11 | | | | | | | | | | | | | | | | PhyPARTID10 | | | | | | | | | | | | | | | |
| PhyPARTID9 | | | | | | | | | | | | | | | | PhyPARTID8 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PhyPARTID11, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 11. PhyPARTID11 gives the mapping of virtual PARTID 11 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID10, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 10. PhyPARTID10 gives the mapping of virtual PARTID 10 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID9, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 9. PhyPARTID9 gives the mapping of virtual PARTID 9 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID8, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 8. PhyPARTID8 gives the mapping of virtual PARTID 8 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM2_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAMVPM2_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x950];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return MPAMVPM2_EL2;
    elsif PSTATE.EL == EL3 then
        return MPAMVPM2_EL2;

```

MSR MPAMVPM2_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x950] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAMVPM2_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        MPAMVPM2_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM3_EL2, MPAM Virtual PARTID Mapping Register 3

The MPAMVPM3_EL2 characteristics are:

Purpose

MPAMVPM3_EL2 provides mappings from virtual PARTIDs 12 - 15 to physical PARTIDs.

[MPAMIDR_EL1.VPMR_MAX](#) field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If [MPAMIDR_EL1.VPMR_MAX](#) == 0, there is only a single MPAMVPM<n>_EL2 register, [MPAMVPM0_EL2](#).

Virtual PARTID mapping is enabled by [MPAMHCR_EL2.EL1_VPMEN](#) for PARTIDs in [MPAM1_EL1](#) and by [MPAMHCR_EL2.EL0_VPMEN](#) for PARTIDs in [MPAM0_EL1](#).

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the [MPAMVPMV_EL2.VPM_V](#) bit in bit position n is set to 1.

Configuration

This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 2. Otherwise, direct accesses to MPAMVPM3_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM3_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| PhyPARTID15 | | | | | | | | | | | | | | | | PhyPARTID14 | | | | | | | | | | | | | | | |
| PhyPARTID13 | | | | | | | | | | | | | | | | PhyPARTID12 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PhyPARTID15, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 15. PhyPARTID15 gives the mapping of virtual PARTID 15 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID14, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 14. PhyPARTID14 gives the mapping of virtual PARTID 14 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID13, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 13. PhyPARTID13 gives the mapping of virtual PARTID 13 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID12, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 12. PhyPARTID12 gives the mapping of virtual PARTID 12 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM3_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAMVPM3_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x958];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            end
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        end
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        end
    else
        return MPAMVPM3_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPM3_EL2;

```

MSR MPAMVPM3_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x958] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAMVPM3_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        MPAMVPM3_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM4_EL2, MPAM Virtual PARTID Mapping Register 4

The MPAMVPM4_EL2 characteristics are:

Purpose

MPAMVPM4_EL2 provides mappings from virtual PARTIDs 16 - 19 to physical PARTIDs.

[MPAMIDR_EL1.VPMR_MAX](#) field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If [MPAMIDR_EL1.VPMR_MAX](#) == 0, there is only a single MPAMVPM<n>_EL2 register, [MPAMVPM0_EL2](#).

Virtual PARTID mapping is enabled by [MPAMHCR_EL2.EL1_VPMEN](#) for PARTIDs in [MPAM1_EL1](#) and by [MPAMHCR_EL2.EL0_VPMEN](#) for PARTIDs in [MPAM0_EL1](#).

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the [MPAMVPMV_EL2.VPM_V](#) bit in bit position n is set to 1.

Configuration

This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 3. Otherwise, direct accesses to MPAMVPM4_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM4_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| PhyPARTID19 | | | | | | | | | | | | | | | | PhyPARTID18 | | | | | | | | | | | | | | | |
| PhyPARTID17 | | | | | | | | | | | | | | | | PhyPARTID16 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PhyPARTID19, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 19. PhyPARTID19 gives the mapping of virtual PARTID 19 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID18, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 18. PhyPARTID18 gives the mapping of virtual PARTID 18 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID17, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 17. PhyPARTID17 gives the mapping of virtual PARTID 17 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID16, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 16. PhyPARTID16 gives the mapping of virtual PARTID 16 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM4_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAMVPM4_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x960];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return MPAMVPM4_EL2;
    elsif PSTATE.EL == EL3 then
        return MPAMVPM4_EL2;

```

MSR MPAMVPM4_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x960] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAMVPM4_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        MPAMVPM4_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM5_EL2, MPAM Virtual PARTID Mapping Register 5

The MPAMVPM5_EL2 characteristics are:

Purpose

MPAMVPM5_EL2 provides mappings from virtual PARTIDs 20 - 23 to physical PARTIDs.

[MPAMIDR_EL1.VPMR_MAX](#) field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If [MPAMIDR_EL1.VPMR_MAX](#) == 0, there is only a single MPAMVPM<n>_EL2 register, [MPAMVPM0_EL2](#).

Virtual PARTID mapping is enabled by [MPAMHCR_EL2.EL1_VPMEN](#) for PARTIDs in [MPAM1_EL1](#) and by [MPAMHCR_EL2.EL0_VPMEN](#) for PARTIDs in [MPAM0_EL1](#).

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the [MPAMVPMV_EL2.VPM_V](#) bit in bit position n is set to 1.

Configuration

This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 4. Otherwise, direct accesses to MPAMVPM5_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM5_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| PhyPARTID23 | | | | | | | | | | | | | | | | PhyPARTID22 | | | | | | | | | | | | | | | |
| PhyPARTID21 | | | | | | | | | | | | | | | | PhyPARTID20 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PhyPARTID23, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 23. PhyPARTID23 gives the mapping of virtual PARTID 23 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID22, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 22. PhyPARTID22 gives the mapping of virtual PARTID 22 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID21, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 21. PhyPARTID21 gives the mapping of virtual PARTID 21 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID20, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 20. PhyPARTID20 gives the mapping of virtual PARTID 20 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM5_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAMVPM5_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x968];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            end
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        end
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        end
    else
        return MPAMVPM5_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPM5_EL2;

```

MSR MPAMVPM5_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x968] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAMVPM5_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        MPAMVPM5_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM6_EL2, MPAM Virtual PARTID Mapping Register 6

The MPAMVPM6_EL2 characteristics are:

Purpose

MPAMVPM6_EL2 provides mappings from virtual PARTIDs 24 - 27 to physical PARTIDs.

[MPAMIDR_EL1.VPMR_MAX](#) field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If [MPAMIDR_EL1.VPMR_MAX](#) == 0, there is only a single MPAMVPM<n>_EL2 register, [MPAMVPM0_EL2](#).

Virtual PARTID mapping is enabled by [MPAMHCR_EL2.EL1_VPMEN](#) for PARTIDs in [MPAM1_EL1](#) and by [MPAMHCR_EL2.EL0_VPMEN](#) for PARTIDs in [MPAM0_EL1](#).

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the [MPAMVPMV_EL2.VPM_V](#) bit in bit position n is set to 1.

Configuration

This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX > 5. Otherwise, direct accesses to MPAMVPM6_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM6_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| PhyPARTID27 | | | | | | | | | | | | | | | | PhyPARTID26 | | | | | | | | | | | | | | | |
| PhyPARTID25 | | | | | | | | | | | | | | | | PhyPARTID24 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PhyPARTID27, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 27. PhyPARTID27 gives the mapping of virtual PARTID 27 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID26, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 26. PhyPARTID26 gives the mapping of virtual PARTID 26 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID25, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 25. PhyPARTID25 gives the mapping of virtual PARTID 25 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID24, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 24. PhyPARTID24 gives the mapping of virtual PARTID 24 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM6_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAMVPM6_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x970];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return MPAMVPM6_EL2;
    elsif PSTATE.EL == EL3 then
        return MPAMVPM6_EL2;

```

MSR MPAMVPM6_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x970] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAMVPM6_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        MPAMVPM6_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPM7_EL2, MPAM Virtual PARTID Mapping Register 7

The MPAMVPM7_EL2 characteristics are:

Purpose

MPAMVPM7_EL2 provides mappings from virtual PARTIDs 28 - 31 to physical PARTIDs.

[MPAMIDR_EL1.VPMR_MAX](#) field gives the index of the highest implemented MPAMVPM<n>_EL2 registers. VPMR_MAX can be as large as 7 (8 registers) or 32 virtual PARTIDs. If [MPAMIDR_EL1.VPMR_MAX](#) == 0, there is only a single MPAMVPM<n>_EL2 register, [MPAMVPM0_EL2](#).

Virtual PARTID mapping is enabled by [MPAMHCR_EL2.EL1_VPMEN](#) for PARTIDs in [MPAM1_EL1](#) and by [MPAMHCR_EL2.EL0_VPMEN](#) for [MPAM0_EL1](#).

A virtual-to-physical PARTID mapping entry, PhyPARTID<n>, is valid only when the [MPAMVPMV_EL2.VPM_V](#) bit in bit position n is set to 1.

Configuration

This register is present only when FEAT_MPAM is implemented, MPAMIDR_EL1.HAS_HCR == 1 and MPAMIDR_EL1.VPMR_MAX == 7. Otherwise, direct accesses to MPAMVPM7_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPM7_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| PhyPARTID31 | | | | | | | | | | | | | | | | PhyPARTID30 | | | | | | | | | | | | | | | |
| PhyPARTID29 | | | | | | | | | | | | | | | | PhyPARTID28 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PhyPARTID31, bits [63:48]

Virtual PARTID Mapping Entry for virtual PARTID 31. PhyPARTID31 gives the mapping of virtual PARTID 31 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID30, bits [47:32]

Virtual PARTID Mapping Entry for virtual PARTID 30. PhyPARTID30 gives the mapping of virtual PARTID 30 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID29, bits [31:16]

Virtual PARTID Mapping Entry for virtual PARTID 29. PhyPARTID29 gives the mapping of virtual PARTID 29 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PhyPARTID28, bits [15:0]

Virtual PARTID Mapping Entry for virtual PARTID 28. PhyPARTID28 gives the mapping of virtual PARTID 28 to a physical PARTID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPM7_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAMVPM7_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x978];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            end
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
        end
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        end
    else
        return MPAMVPM7_EL2;
elsif PSTATE.EL == EL3 then
    return MPAMVPM7_EL2;

```

MSR MPAMVPM7_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x978] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAMVPM7_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        MPAMVPM7_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMVPMV_EL2, MPAM Virtual Partition Mapping Valid Register

The MPAMVPMV_EL2 characteristics are:

Purpose

Valid bits for virtual PARTID mapping entries. Each bit m corresponds to virtual PARTID mapping entry m in the MPAMVPM<n>_EL2 registers where n = m >> 2.

Configuration

This register is present only when FEAT_MPAM is implemented and MPAMIDR_EL1.HAS_HCR == 1. Otherwise, direct accesses to MPAMVPMV_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

MPAMVPMV_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | |
| VPM_V31 | VPM_V30 | VPM_V29 | VPM_V28 | VPM_V27 | VPM_V26 | VPM_V25 | VPM_V24 | VPM_V23 | VPM_V22 | VPM_V21 | VPM_V20 | VPM_V19 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 |

Bits [63:32]

Reserved, RES0.

VPM_V<m>, bit [m], for m = 31 to 0

Contains valid bit for virtual PARTID mapping entry corresponding to virtual PARTID<m>.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMVPMV_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPAMVPMV_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x938];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return MPAMVPMV_EL2;
    elsif PSTATE.EL == EL3 then
        return MPAMVPMV_EL2;

```

MSR MPAMVPMV_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1010 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x938] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && MPAM3_EL3.TRAPLOWER == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            MPAMVPMV_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        MPAMVPMV_EL2 = X[t];

```

MPIDR_EL1, Multiprocessor Affinity Register

The MPIDR_EL1 characteristics are:

Purpose

In a multiprocessor system, provides an additional PE identification mechanism for scheduling purposes.

Configuration

AArch64 System register MPIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [MPIDR\[31:0\]](#).

In a uniprocessor system, Arm recommends that each Aff<n> field of this register returns a value of 0.

Attributes

MPIDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|------|----|----|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|------|------|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | Aff3 | | | | | | | | |
| RES1 | | U | RES0 | | | | | MT | Aff2 | | | | | | | | Aff1 | | | | | | | | Aff0 | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3. See the description of Aff0 for more information.

Aff3 is not supported in AArch32 state.

Bit [31]

Reserved, RES1.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

| U | Meaning |
|-----|---|
| 0b0 | Processor is part of a multiprocessor system. |
| 0b1 | Processor is part of a uniprocessor system. |

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using a multithreading type approach. See the description of Aff0 for more information about affinity levels.

| MT | Meaning |
|-----|---|
| 0b0 | Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent. |
| 0b1 | Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent. |

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

Aff0, bits [7:0]

Affinity level 0. This is the affinity level that is most significant for determining PE behavior. Higher affinity levels are increasingly less significant in determining PE behavior. The assigned value of the MPIDR.{Aff2, Aff1, Aff0} or [MPIDR_EL1](#).{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

Accessing MPIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MPIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.MPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() then
        return VMPIDR_EL2;
    else
        return MPIDR_EL1;
elseif PSTATE.EL == EL2 then
    return MPIDR_EL1;
elseif PSTATE.EL == EL3 then
    return MPIDR_EL1;

```

MVFR0_EL1, AArch32 Media and VFP Feature Register 0

The MVFR0_EL1 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and Floating-point implementation.

Must be interpreted with [MVFR1_EL1](#) and [MVFR2_EL1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register MVFR0_EL1 bits [31:0] are architecturally mapped to AArch32 System register [MVFR0\[31:0\]](#).

In an implementation where at least one Exception level supports execution in AArch32 state, but there is no support for Advanced SIMD and floating-point operation, this register is RAZ.

Attributes

MVFR0_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|---------|----|----|----|--------|----|----|----|----------|----|----|----|--------|----|----|----|------|----|----|----|------|----|----|----|---------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| FPRound | | | | FPShVec | | | | FPSqrt | | | | FPDivide | | | | FPTrap | | | | FPDP | | | | FPSP | | | | SIMDReg | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

FPRound, bits [31:28]

Floating-Point Rounding modes. Indicates whether the floating-point implementation provides support for rounding modes. Defined values are:

| FPRound | Meaning |
|---------|--|
| 0b0000 | Not implemented, or only Round to Nearest mode supported, except that Round towards Zero mode is supported for VCVT instructions that always use that rounding mode regardless of the FPSCR setting. |
| 0b0001 | All rounding modes supported. |

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

FPSHVec, bits [27:24]

Short Vectors. Indicates whether the floating-point implementation provides support for the use of short vectors. Defined values are:

| FPSHVec | Meaning |
|----------------|-----------------------------------|
| 0b0000 | Short vectors not supported. |
| 0b0001 | Short vector operation supported. |

All other values are reserved.

In Armv8-A the only permitted value is 0b0000.

FPSqrt, bits [23:20]

Square Root. Indicates whether the floating-point implementation provides support for the ARMv6 VFP square root operations. Defined values are:

| FPSqrt | Meaning |
|---------------|----------------------------|
| 0b0000 | Not supported in hardware. |
| 0b0001 | Supported. |

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

The VSQRT.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VSQRT.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

FPDivide, bits [19:16]

Indicates whether the floating-point implementation provides support for VFP divide operations. Defined values are:

| FPDivide | Meaning |
|-----------------|----------------------------|
| 0b0000 | Not supported in hardware. |
| 0b0001 | Supported. |

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0001.

The VDIV.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VDIV.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

FPTrap, bits [15:12]

Floating Point Exception Trapping. Indicates whether the floating-point implementation provides support for exception trapping. Defined values are:

| FPTrap | Meaning |
|---------------|----------------|
| 0b0000 | Not supported. |
| 0b0001 | Supported. |

All other values are reserved.

A value of 0b0001 indicates that, when the corresponding trap is enabled, a floating-point exception generates an exception.

FPDP, bits [11:8]

Double Precision. Indicates whether the floating-point implementation provides support for double-precision operations. Defined values are:

| FPDP | Meaning |
|--------|--|
| 0b0000 | Not supported in hardware. |
| 0b0001 | Supported, VFPv2. |
| 0b0010 | Supported, VFPv3, VFPv4, or Armv8. VFPv3 and Armv8 add an instruction to load a double-precision floating-point constant, and conversions between double-precision and fixed-point values. |

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

A value of 0b0001 or 0b0010 indicates support for all VFP double-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F64 is only available if the Square root field is 0b0001.
- VDIV.F64 is only available if the Divide field is 0b0001.
- Conversion between double-precision and single-precision is only available if the single-precision field is nonzero.

FPSP, bits [7:4]

Single Precision. Indicates whether the floating-point implementation provides support for single-precision operations. Defined values are:

| FPSP | Meaning |
|--------|---|
| 0b0000 | Not supported in hardware. |
| 0b0001 | Supported, VFPv2. |
| 0b0010 | Supported, VFPv3 or VFPv4. VFPv3 adds an instruction to load a single-precision floating-point constant, and conversions between single-precision and fixed-point values. |

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

A value of 0b0001 or 0b0010 indicates support for all VFP single-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F32 is only available if the Square root field is 0b0001.
- VDIV.F32 is only available if the Divide field is 0b0001.
- Conversion between double-precision and single-precision is only available if the double-precision field is nonzero.

SIMDReg, bits [3:0]

Advanced SIMD registers. Indicates whether the Advanced SIMD and floating-point implementation provides support for the Advanced SIMD and floating-point register bank. Defined values are:

| SIMDReg | Meaning |
|---------|--|
| 0b0000 | The implementation has no Advanced SIMD and floating-point support. |
| 0b0001 | The implementation includes floating-point support with 16 x 64-bit registers. |
| 0b0010 | The implementation includes Advanced SIMD and floating-point support with 32 x 64-bit registers. |

All other values are reserved.

In Armv8-A the permitted values are 0b0000 and 0b0010.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing MVFR0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MVFR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return MVFR0_EL1;
elseif PSTATE.EL == EL2 then
    return MVFR0_EL1;
elseif PSTATE.EL == EL3 then
    return MVFR0_EL1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MVFR1_EL1, AArch32 Media and VFP Feature Register 1

The MVFR1_EL1 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and Floating-point implementation.

Must be interpreted with [MVFR0_EL1](#) and [MVFR2_EL1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register MVFR1_EL1 bits [31:0] are architecturally mapped to AArch32 System register [MVFR1\[31:0\]](#).

In an implementation where at least one Exception level supports execution in AArch32 state, but there is no support for Advanced SIMD and floating-point operation, this register is RAZ.

Attributes

MVFR1_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|------|----|----|----|--------|----|----|----|--------|----|----|----|---------|----|----|----|--------|----|----|----|--------|----|----|----|-------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| SIMDFMAC | | | | FPHP | | | | SIMDHP | | | | SIMDSP | | | | SIMDInt | | | | SIMDLS | | | | FPDNaN | | | | FPFtZ | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

SIMDFMAC, bits [31:28]

Advanced SIMD Fused Multiply-Accumulate. Indicates whether the Advanced SIMD implementation provides fused multiply accumulate instructions. Defined values are:

| SIMDFMAC | Meaning |
|----------|------------------|
| 0b0000 | Not implemented. |
| 0b0001 | Implemented. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

The Advanced SIMD and floating-point implementations must provide the same level of support for these instructions.

FPHP, bits [27:24]

Floating Point Half Precision. Indicates the level of half-precision floating-point support. Defined values are:

| FPHP | Meaning |
|-------------|---|
| 0b0000 | Not supported. |
| 0b0001 | Floating-point half-precision conversion instructions are supported for conversion between single-precision and half-precision. |
| 0b0010 | As for 0b0001, and adds instructions for conversion between double-precision and half-precision. |
| 0b0011 | As for 0b0010, and adds support for half-precision floating-point arithmetic. |

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 in an implementation without floating-point support.
- 0b0010 in an implementation with floating-point support that does not include the FEAT_FP16 extension.
- 0b0011 in an implementation with floating-point support that includes the FEAT_FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the SIMDHP field, meaning the permitted values are:

| Half Precision instructions supported | FPHP | SIMDHP |
|--|-------------|---------------|
| No support | 0b0000 | 0b0000 |
| Conversions only | 0b0010 | 0b0001 |
| Conversions and arithmetic | 0b0011 | 0b0010 |

SIMDHP, bits [23:20]

Advanced SIMD Half Precision. Indicates the level of half-precision floating-point support. Defined values are:

| SIMDHP | Meaning |
|---------------|---|
| 0b0000 | Not supported. |
| 0b0001 | SIMD half-precision conversion instructions are supported for conversion between single-precision and half-precision. |
| 0b0010 | As for 0b0001, and adds support for half-precision floating-point arithmetic. |

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 in an implementation without SIMD floating-point support.
- 0b0001 in an implementation with SIMD floating-point support that does not include the FEAT_FP16 extension.
- 0b0010 in an implementation with SIMD floating-point support that includes the FEAT_FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the FPHP field, meaning the permitted values are:

| Half Precision instructions supported | FPHP | SIMDHP |
|--|-------------|---------------|
| No support | 0b0000 | 0b0000 |
| Conversions only | 0b0010 | 0b0001 |
| Conversions and arithmetic | 0b0011 | 0b0010 |

SIMDSP, bits [19:16]

Advanced SIMD Single Precision. Indicates whether the Advanced SIMD and floating-point implementation provides single-precision floating-point instructions. Defined values are:

| SIMDSP | Meaning |
|---------------|---|
| 0b0000 | Not implemented. |
| 0b0001 | Implemented. This value is permitted only if the SIMDInt field is 0b0001. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

SIMDInt, bits [15:12]

Advanced SIMD Integer. Indicates whether the Advanced SIMD and floating-point implementation provides integer instructions. Defined values are:

| SIMDInt | Meaning |
|----------------|------------------|
| 0b0000 | Not implemented. |
| 0b0001 | Implemented. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

SIMDLS, bits [11:8]

Advanced SIMD Load/Store. Indicates whether the Advanced SIMD and floating-point implementation provides load/store instructions. Defined values are:

| SIMDLS | Meaning |
|---------------|------------------|
| 0b0000 | Not implemented. |
| 0b0001 | Implemented. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

FPDNAN, bits [7:4]

Default NaN mode. Indicates whether the floating-point implementation provides support only for the Default NaN mode. Defined values are:

| FPDNAN | Meaning |
|---------------|--|
| 0b0000 | Not implemented, or hardware supports only the Default NaN mode. |
| 0b0001 | Hardware supports propagation of NaN values. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

FPFtZ, bits [3:0]

Flush to Zero mode. Indicates whether the floating-point implementation provides support only for the Flush-to-Zero mode of operation. Defined values are:

| FPFtZ | Meaning |
|--------------|---|
| 0b0000 | Not implemented, or hardware supports only the Flush-to-Zero mode of operation. |
| 0b0001 | Hardware supports full denormalized number arithmetic. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing MVFR1_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MVFR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID3 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return MVFR1_EL1;
elseif PSTATE.EL == EL2 then
    return MVFR1_EL1;
elseif PSTATE.EL == EL3 then
    return MVFR1_EL1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MVFR2_EL1, AArch32 Media and VFP Feature Register 2

The MVFR2_EL1 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and Floating-point implementation.

Must be interpreted with [MVFR0_EL1](#) and [MVFR1_EL1](#).

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch64 System register MVFR2_EL1 bits [31:0] are architecturally mapped to AArch32 System register [MVFR2\[31:0\]](#).

In an implementation where at least one Exception level supports execution in AArch32 state, but there is no support for Advanced SIMD and floating-point operation, this register is RAZ.

Attributes

MVFR2_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | FPMisc | | | | SIMDMisc | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:8]

Reserved, RES0.

FPMisc, bits [7:4]

Indicates whether the floating-point implementation provides support for miscellaneous VFP features.

| FPMisc | Meaning |
|--------|---|
| 0b0000 | Not implemented, or no support for miscellaneous features. |
| 0b0001 | Support for Floating-point selection. |
| 0b0010 | As 0b0001, and Floating-point Conversion to Integer with Directed Rounding modes. |
| 0b0011 | As 0b0010, and Floating-point Round to Integer Floating-point. |
| 0b0100 | As 0b0011, and Floating-point MaxNum and MinNum. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0100.

SIMDMisc, bits [3:0]

Indicates whether the Advanced SIMD implementation provides support for miscellaneous Advanced SIMD features.

| SIMDMisc | Meaning |
|----------|--|
| 0b0000 | Not implemented, or no support for miscellaneous features. |
| 0b0001 | Floating-point Conversion to Integer with Directed Rounding modes. |
| 0b0010 | As 0b0001, and Floating-point Round to Integer Floating-point. |
| 0b0011 | As 0b0010, and Floating-point MaxNum and MinNum. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0011.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | UNKNOWN | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, UNKNOWN.

Accessing MVFR2_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, MVFR2_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID3 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return MVFR2_EL1;
    elsif PSTATE.EL == EL2 then
        return MVFR2_EL1;
    elsif PSTATE.EL == EL3 then
        return MVFR2_EL1;

```

NZCV, Condition Flags

The NZCV characteristics are:

Purpose

Allows access to the condition flags.

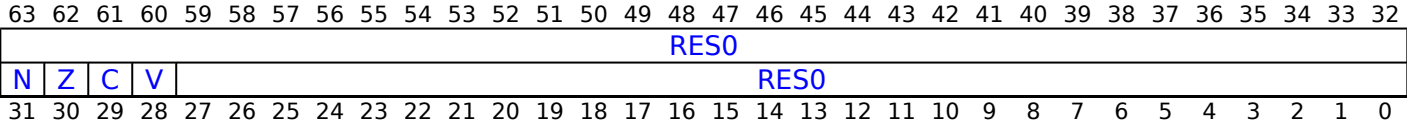
Configuration

There are no configuration notes.

Attributes

NZCV is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative condition flag. Set to 1 if the result of the last flag-setting instruction was negative.

Z, bit [30]

Zero condition flag. Set to 1 if the result of the last flag-setting instruction was zero, and to 0 otherwise. A result of zero often indicates an equal result from a comparison.

C, bit [29]

Carry condition flag. Set to 1 if the last flag-setting instruction resulted in a carry condition, for example an unsigned overflow on an addition.

V, bit [28]

Overflow condition flag. Set to 1 if the last flag-setting instruction resulted in an overflow condition, for example a signed overflow on an addition.

Bits [27:0]

Reserved, RES0.

Accessing NZCV

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, NZCV

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    return Zeros(32):PSTATE.<N,Z,C,V>:Zeros(28);
elsif PSTATE.EL == EL1 then
    return Zeros(32):PSTATE.<N,Z,C,V>:Zeros(28);
elsif PSTATE.EL == EL2 then
    return Zeros(32):PSTATE.<N,Z,C,V>:Zeros(28);
elsif PSTATE.EL == EL3 then
    return Zeros(32):PSTATE.<N,Z,C,V>:Zeros(28);

```

MSR NZCV, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    PSTATE.<N,Z,C,V> = X[t]<31:28>;
elsif PSTATE.EL == EL1 then
    PSTATE.<N,Z,C,V> = X[t]<31:28>;
elsif PSTATE.EL == EL2 then
    PSTATE.<N,Z,C,V> = X[t]<31:28>;
elsif PSTATE.EL == EL3 then
    PSTATE.<N,Z,C,V> = X[t]<31:28>;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSDLR_EL1, OS Double Lock Register

The OSDLR_EL1 characteristics are:

Purpose

Used to control the OS Double Lock.

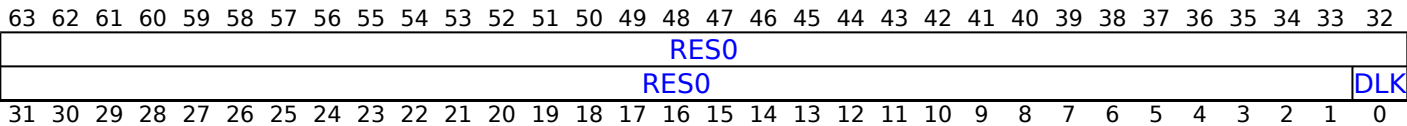
Configuration

AArch64 System register OSDLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGOSDLR\[31:0\]](#).

Attributes

OSDLR_EL1 is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

DLK, bit [0]

When FEAT_DoubleLock is implemented:

OS Double Lock control bit.

| DLK | Meaning |
|-----|--|
| 0b0 | OS Double Lock unlocked. |
| 0b1 | OS Double Lock locked, if DBGPRCR_EL1 .CORENPDRQ (Core no powerdown request) bit is set to 0 and the PE is in Non-debug state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RAZ/WI.

Accessing OSDLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, OSDLR_EL1

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0001 | 0b0011 | 0b100 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) ||
boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
IsFeatureImplemented(FEAT_DoubleLock) && HDFGRTR_EL2.OSDLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' && (IsFeatureImplemented(FEAT_DoubleLock)
|| boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL2.TDOSA") then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) ||
boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return OSDLR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) ||
boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) ||
boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return OSDLR_EL1;
elsif PSTATE.EL == EL3 then
    return OSDLR_EL1;

```

MSR OSDLR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0001 | 0b0011 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) ||
boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
IsFeatureImplemented(FEAT_DoubleLock) && HDFGWTR_EL2.OSDLR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' && (IsFeatureImplemented(FEAT_DoubleLock)
|| boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL2.TDOSA") then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) ||
boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            OSDLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) ||
boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' && (IsFeatureImplemented(FEAT_DoubleLock) ||
boolean IMPLEMENTATION_DEFINED "Trapped by MDCR_EL3.TDOSA") then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            OSDLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    OSDLR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSDTRRX_EL1, OS Lock Data Transfer Register, Receive

The OSDTRRX_EL1 characteristics are:

Purpose

Used for save and restore of [DBGDTRRX_EL0](#). It is a component of the Debug Communications Channel.

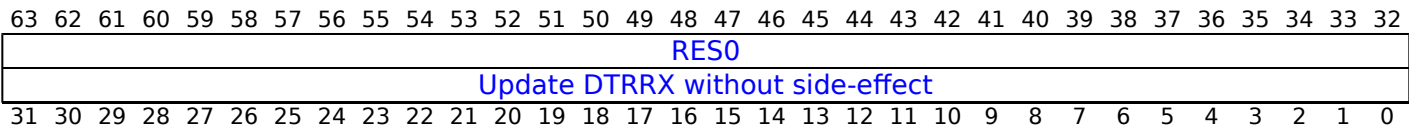
Configuration

AArch64 System register OSDTRRX_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGDTRRXext\[31:0\]](#).

Attributes

OSDTRRX_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

Bits [31:0]

- Update DTRRX without side-effect.
- Writes to this register update the value in DTRRX and do not change RXfull.
- Reads of this register return the last value written to DTRRX and do not change RXfull.
- For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

Accessing OSDTRRX_EL1

Arm deprecates reads and writes of OSDTRRX_EL1 when the OS Lock is unlocked.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, OSDTRRX_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    return OSDTRRX_EL1;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return OSDTRRX_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return OSDTRRX_EL1;
elsif PSTATE.EL == EL3 then
    return OSDTRRX_EL1;

```

MSR OSDTRRX_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    OSDTRRX_EL1 = X[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDTRRX_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDTRRX_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    OSDTRRX_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSDTRTX_EL1, OS Lock Data Transfer Register, Transmit

The OSDTRTX_EL1 characteristics are:

Purpose

Used for save/restore of [DBGDTRTX_EL0](#). It is a component of the Debug Communications Channel.

Configuration

AArch64 System register OSDTRTX_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGDTRTXext\[31:0\]](#).

Attributes

OSDTRTX_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Return DTRTX without side-effect | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

Bits [31:0]

Return DTRTX without side-effect.

Reads of this register return the value in DTRTX and do not change TXfull.

Writes of this register update the value in DTRTX and do not change TXfull.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

Accessing OSDTRTX_EL1

Arm deprecates reads and writes of OSDTRTX_EL1 when the OS Lock is unlocked.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, OSDTRTX_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    return OSDTRTX_EL1;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return OSDTRTX_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return OSDTRTX_EL1;
elsif PSTATE.EL == EL3 then
    return OSDTRTX_EL1;

```

MSR OSDTRTX_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    OSDTRTX_EL1 = X[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TDCC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDTRTX_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        OSDTRTX_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    OSDTRTX_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSECCR_EL1, OS Lock Exception Catch Control Register

The OSECCR_EL1 characteristics are:

Purpose

Provides a mechanism for an operating system to access the contents of [EDECCR](#) that are otherwise invisible to software, so it can save/restore the contents of [EDECCR](#) over powerdown on behalf of the external debugger.

Configuration

AArch64 System register OSECCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGOSECCR\[31:0\]](#).

AArch64 System register OSECCR_EL1 bits [31:0] are architecturally mapped to External register [EDECCR\[31:0\]](#).

If [OSLSR_EL1.OSLK](#) == 0, then OSECCR_EL1 returns an UNKNOWN value on reads and ignores writes.

Attributes

OSECCR_EL1 is a 64-bit register.

Field descriptions

When OSLSR_EL1.OSLK == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | EDECCR | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

EDECCR, bits [31:0]

Used for save/restore to [EDECCR](#) over powerdown.

Reads or writes to this field are indirect accesses to [EDECCR](#).

Accessing OSECCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, OSECCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | 0b0110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.OSECCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elseif OSLSR_EL1.OSLK == '0' then
        return bits(64) UNKNOWN;
    else
        return OSECCR_EL1;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elseif OSLSR_EL1.OSLK == '0' then
        return bits(64) UNKNOWN;
    else
        return OSECCR_EL1;
elseif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' then
        return bits(64) UNKNOWN;
    else
        return OSECCR_EL1;

```

MSR OSECCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0000 | 0b0110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.OSECCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' then
        //no operation
    else
        OSECCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif OSLSR_EL1.OSLK == '0' then
        //no operation
    else
        OSECCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if OSLSR_EL1.OSLK == '0' then
        //no operation
    else
        OSECCR_EL1 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSLAR_EL1, OS Lock Access Register

The OSLAR_EL1 characteristics are:

Purpose

Used to lock or unlock the OS Lock.

Configuration

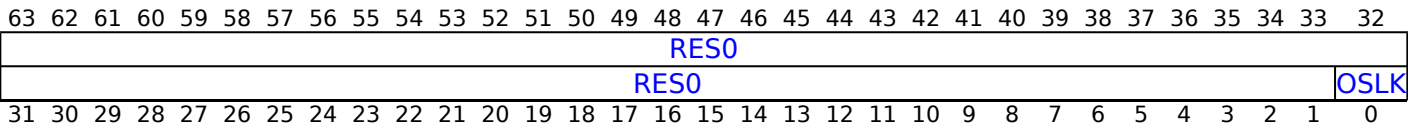
AArch64 System register OSLAR_EL1 bits [31:0] are architecturally mapped to External register [OSLAR_EL1\[31:0\]](#).

The OS Lock can also be locked or unlocked using [DBGOSLAR](#).

Attributes

OSLAR_EL1 is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

OSLK, bit [0]

On writes to OSLAR_EL1, bit[0] is copied to the OS Lock.

Use [OSLSR_EL1](#).OSLK to check the current status of the lock.

Accessing OSLAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MSR OSLAR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0001 | 0b0000 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.OSLAR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            OSLAR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDOSA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                OSLAR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        OSLAR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSLSR_EL1, OS Lock Status Register

The OLSR_EL1 characteristics are:

Purpose

Provides the status of the OS Lock.

Configuration

AArch64 System register OSLSR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGOSLSR\[31:0\]](#).

Attributes

OSLSR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|---------|----|----|-----|------|----|---------|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | OSLM[1] | | | hTT | OSLK | | OSLM[0] | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:4]

Reserved, RES0.

OSLM, bits [3, 0]

OS Lock model implemented. Identifies the form of OS save and restore mechanism implemented.

| OSLM | Meaning |
|------|--------------------------|
| 0b00 | OS Lock not implemented. |
| 0b10 | OS Lock implemented. |

All other values are reserved. In an Armv8 implementation the value 0b00 is not permitted.

The OSLM field is split as follows:

- OSLM[1] is OLSR_EL1[3].
- OSLM[0] is OLSR_EL1[0].

nTT, bit [2]

Not 32-bit access. This bit is always RAZ. It indicates that a 32-bit access is needed to write the key to the OS Lock Access Register.

OSLK, bit [1]

OS Lock Status.

| OSLK | Meaning |
|------|-------------------|
| 0b0 | OS Lock unlocked. |
| 0b1 | OS Lock locked. |

The OS Lock is locked and unlocked by writing to the OS Lock Access Register.

The reset behavior of this field is:

- On a Cold reset, this field resets to 1.

Accessing OSLSR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, OSLSR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b000 | 0b0001 | 0b0001 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.OSLSR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return OSLSR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDOSA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return OSLSR_EL1;
elsif PSTATE.EL == EL3 then
    return OSLSR_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PAN, Privileged Access Never

The PAN characteristics are:

Purpose

Allows access to the Privileged Access Never bit.

Configuration

This register is present only when FEAT_PAN is implemented. Otherwise, direct accesses to PAN are UNDEFINED.

Attributes

PAN is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|-----|----|------|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|--|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | PAN | | RES0 | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | |

Bits [63:23]

Reserved, RES0.

PAN, bit [22]

Privileged Access Never.

| PAN | Meaning |
|-----|--|
| 0b0 | Privileged reads and write are not disabled by this mechanism. |
| 0b1 | Disables privileged read and write accesses to addresses accessible at EL0 for an enabled stage 1 translation regime that defines the EL0 permissions. |

The value of this bit is usually preserved on taking an exception, except in the following situations:

- When the target of the exception is EL1, and the value of the [SCTLR_EL1.SPAN](#) bit is 0, this bit is set to 1.
- When the target of the exception is EL2, [HCR_EL2](#).{E2H, TGE} is {1, 1}, and the value of the [SCTLR_EL2.SPAN](#) bit is 0, this bit is set to 1.

Bits [21:0]

Reserved, RES0.

Accessing PAN

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PAN

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    return Zeros(41):PSTATE.PAN:Zeros(22);
elsif PSTATE.EL == EL2 then
    return Zeros(41):PSTATE.PAN:Zeros(22);
elsif PSTATE.EL == EL3 then
    return Zeros(41):PSTATE.PAN:Zeros(22);

```

MSR PAN, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    PSTATE.PAN = X[t]<22>;
elsif PSTATE.EL == EL2 then
    PSTATE.PAN = X[t]<22>;
elsif PSTATE.EL == EL3 then
    PSTATE.PAN = X[t]<22>;

```

MSR PAN, #<imm>

| op0 | op1 | CRn | op2 |
|------|-------|--------|-------|
| 0b00 | 0b000 | 0b0100 | 0b100 |

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PAR_EL1, Physical Address Register

The PAR_EL1 characteristics are:

Purpose

Returns the output address (OA) from an Address translation instruction that executed successfully, or fault information if the instruction did not execute successfully.

Configuration

AArch64 System register PAR_EL1 bits [63:0] are architecturally mapped to AArch32 System register [PAR\[63:0\]](#).

Attributes

PAR_EL1 is a 64-bit register.

Field descriptions

When PAR_EL1.F == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|------|----|----|----|-----------|----|----|----|-----------|----|---------------------------|----|----|--|----|--|----|----|------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | | 42 | | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ATTR | | | | | | | | RES0 | | | | PA[51:48] | | | | PA[47:12] | | | | | | | | | | | | | | | | | |
| PA[47:12] | | | | | | | | | | | | | | | | NSE | | IMPLEMENTATION DEFINED | | | | NS | | SH | | RES0 | | | | | | F | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | | 10 | | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR_EL1 can return a value that indicates the resulting attributes, rather than the values that appear in the translation table descriptors. More precisely:

- The PAR_EL1.{ATTR, SH} fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the translation table descriptors.
- See the PAR_EL1.NS bit description for constraints on the value it returns.

ATTR, bits [63:56]

Memory attributes for the returned output address. This field uses the same encoding as the Attr<n> fields in [MAIR_EL1](#), [MAIR_EL2](#), and [MAIR_EL3](#).

The value returned in this field can be the resulting attribute that is actually implemented by the implementation, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

Note

The attributes presented are consistent with the stages of translation applied in the address translation instruction. If the instruction performed a stage 1 translation only, the attributes are from the stage 1 translation. If the instruction performed a stage 1 and stage 2 translation, the attributes are from the combined stage 1 and stage 2 translation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52]

Reserved, RES0.

PA[51:48], bits [51:48]

When FEAT_LPA is implemented:

Extension to PA[47:12]. For more information, see PA[47:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PA[47:12], bits [47:12]

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[47:12].

When FEAT_LPA is implemented and 52-bit addresses are in use, PA[51:48] forms the upper part of the address value. Otherwise, when 52-bit addresses are not in use, PA[51:48] is RES0.

For implementations with fewer than 48 physical address bits, the corresponding upper bits in this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSE, bit [11]

When FEAT_RME is implemented:

Reports the NSE attribute for a translation table entry from the EL3 translation regime.

For a description of the values derived by evaluating NS and NSE together, see PAR_EL1.NS.

For a result from a Secure, Non-secure, or Realm translation regime, this bit is UNKNOWN.

Otherwise:

Reserved, RES1.

IMPLEMENTATION DEFINED, bit [10]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [9]

When FEAT_RME is implemented:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime, a Realm translation regime, and the EL3 translation regime.

For a result from an EL3 translation regime, NS and NSE are evaluated together to report the physical address space:

| NSE | NS | Meaning |
|-----|-----|-------------|
| 0b0 | 0b0 | Secure. |
| 0b0 | 0b1 | Non-secure. |
| 0b1 | 0b0 | Root. |
| 0b1 | 0b1 | Realm. |

For a result from a Secure translation regime, when [SCR_EL3.EEL2](#) is 1, this bit reflects the Security state of the intermediate physical address space of the translation for the instructions:

- In AArch64 state: [AT S1E1R](#), [AT S1E1W](#), [AT S1E1RP](#), [AT S1E1WP](#), [AT S1E0R](#), and [AT S1E0W](#).
- In AArch32 state: [ATS1CPR](#), [ATS1CPW](#), [ATS1CPRP](#), [ATS1CPWP](#), [ATS1CUR](#), and [ATS1CUW](#).

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

For a result from an S1E1 or S1E0 operation on the Realm EL1&0 translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, when [SCR_EL3.EEL2](#) is 1, this bit reflects the Security state of the intermediate physical address space of the translation for the instructions:

- In AArch64 state: [AT S1E1R](#), [AT S1E1W](#), [AT S1E1RP](#), [AT S1E1WP](#), [AT S1E0R](#), and [AT S1E0W](#).
- In AArch32 state: [ATS1CPR](#), [ATS1CPW](#), [ATS1CPRP](#), [ATS1CPWP](#), [ATS1CUR](#), and [ATS1CUW](#).

Otherwise, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH, bits [8:7]

Shareability attribute, for the returned output address.

| SH | Meaning |
|------|------------------|
| 0b00 | Non-shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

The value 0b01 is reserved.

Note

This field returns the value 0b10 for:

- Any type of Device memory.
 - Normal memory with both Inner Non-cacheable and Outer Non-cacheable attributes.
-

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:1]

Reserved, RES0.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

| F | Meaning |
|-----|---|
| 0b0 | Address translation completed successfully. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When PAR_EL1.F == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------------|----|----|----|----|----|----|----|---------------------------|----|----|----|---------------------------|----|----|----|------|----|------|----|-----|----|------|----|-----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | IMPLEMENTATION DEFINED | | | | IMPLEMENTATION DEFINED | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | RES1 | | RES0 | S | PTW | | RES0 | | FST | | | | F | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

This section describes the register value returned by a fault on the execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

IMPLEMENTATION DEFINED, bits [63:56]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [55:52]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [51:48]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [47:12]

Reserved, RES0.

Bit [11]

Reserved, RES1.

Bit [10]

Reserved, RES0.

S, bit [9]

Indicates the translation stage at which the translation aborted:

| S | Meaning |
|----------|--|
| 0b0 | Translation aborted because of a fault in the stage 1 translation. |
| 0b1 | Translation aborted because of a fault in the stage 2 translation. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PTW, bit [8]

If this bit is set to 1, it indicates the translation aborted because of a stage 2 fault during a stage 1 translation table walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

FST, bits [6:1]

Fault status code, as shown in the Data Abort ESR encoding.

| FST | Meaning | Applies when |
|------------|---|---|
| 0b000000 | Address size fault, level 0 of translation or translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000100 | Translation fault, level 0. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001000 | Access flag fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001100 | Permission fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010011 | Synchronous External abort on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented |
| 0b010100 | Synchronous External abort on translation table walk or hardware update of translation table, level 0. | |
| 0b010101 | Synchronous External abort on translation table walk or hardware update of translation table, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk or hardware update of translation table, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk or hardware update of translation table, level 3. | |
| 0b011011 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented and FEAT_RAS is not implemented |
| 0b011100 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 0. | When FEAT_RAS is not implemented |
| 0b011101 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 1. | When FEAT_RAS is not implemented |
| 0b011110 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 2. | When FEAT_RAS is not implemented |
| 0b011111 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level 3. | When FEAT_RAS is not implemented |
| 0b100011 | Granule Protection Fault on translation table walk or hardware update of translation table, level -1. | When FEAT_RME is implemented and FEAT_LPA2 is implemented |
| 0b100100 | Granule Protection Fault on translation table walk or hardware update of translation table, level 0. | When FEAT_RME is implemented |
| 0b100101 | Granule Protection Fault on translation table walk or hardware update of translation table, level 1. | When FEAT_RME is implemented |
| 0b100110 | Granule Protection Fault on translation table walk or hardware update of translation table, level 2. | When FEAT_RME is implemented |

| | | |
|----------|---|--------------------------------------|
| 0b100111 | Granule Protection Fault on translation table walk or hardware update of translation table, level 3. | When FEAT_RME is implemented |
| 0b101000 | Granule Protection Fault, not on translation table walk or hardware update of translation table. | When FEAT_RME is implemented |
| 0b101001 | Address size fault, level -1. | When FEAT_LPA2 is implemented |
| 0b101011 | Translation fault, level -1. | When FEAT_LPA2 is implemented |
| 0b110000 | TLB conflict abort. | |
| 0b110001 | Unsupported atomic hardware update fault. | When FEAT_HAFDBS is implemented |
| 0b111101 | Section Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format. | When EL1 is capable of using AArch32 |
| 0b111110 | Page Domain fault, from an AArch32 stage 1 EL1&0 translation regime using Short-descriptor translation table format. | When EL1 is capable of using AArch32 |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

| F | Meaning |
|-----|------------------------------|
| 0b1 | Address translation aborted. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PAR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0111 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.PAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return PAR_EL1;
elsif PSTATE.EL == EL2 then
    return PAR_EL1;
elsif PSTATE.EL == EL3 then
    return PAR_EL1;

```

MSR PAR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0111 | 0b0100 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.PAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        PAR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    PAR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PAR_EL1 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMBIDR_EL1, Profiling Buffer ID Register

The PMBIDR_EL1 characteristics are:

Purpose

Provides information to software as to whether the buffer can be programmed at the current Exception level.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMBIDR_EL1 are UNDEFINED.

Attributes

PMBIDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|-------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | EA | | | | RES0 | | F | P | Align | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:12]

Reserved, RES0.

EA, bits [11:8]

External Abort handling. Describes how the PE manages External aborts on writes made by the Statistical Profiling Extension to the Profiling Buffer.

| EA | Meaning |
|--------|---|
| 0b0000 | Not described. |
| 0b0001 | The PE ignores External aborts on writes made by the Statistical Profiling Extension. |
| 0b0010 | The External abort generates an SError interrupt at the PE. |

All other values are reserved.

From Armv8.8, the value 0b0000 is not permitted.

Access to this field is **RO**.

Bits [7:6]

Reserved, RES0.

F, bit [5]

Flag updates. Describes how address translations performed by the Statistical Profiling Extension manage the Access flag and dirty state.

| F | Meaning |
|----------|--|
| 0b0 | Hardware management of the Access flag and dirty state for accesses made by the Statistical Profiling Extension is always disabled for all translation stages. |
| 0b1 | Hardware management of the Access flag and dirty state for accesses made by the Statistical Profiling Extension is controlled in the same way as explicit memory accesses in the Profiling Buffer owning translation regime. |

Note

If hardware management of the Access flag is disabled for a stage of translation, an access to a Page or Block with the Access flag bit not set in the descriptor will generate an Access Flag fault.

If hardware management of the dirty state is disabled for a stage of translation, an access to a Page or Block will ignore the Dirty Bit Modifier in the descriptor and might generate a Permission fault, depending on the values of the access permission bits in the descriptor.

From Armv8.8, the value 0 is not permitted.

Access to this field is **RO**.

P, bit [4]

Programming not allowed. When read at EL3, this field reads as zero. Otherwise, indicates that the Profiling Buffer is owned by a higher Exception level or another Security state. Defined values are:

| P | Meaning |
|----------|--------------------------|
| 0b0 | Programming is allowed. |
| 0b1 | Programming not allowed. |

The value read from this field depends on the current Exception level and the Effective values of [MDCR_EL3.NSPB](#), [MDCR_EL3.NSPBE](#), and [MDCR_EL2.E2PB](#):

- If EL3 is implemented, and the owning Security state is Secure state, this field reads as one from:
 - Non-secure EL1 and Non-secure EL2.
 - If FEAT_RME is implemented, Realm EL1 and Realm EL2.
 - If Secure EL2 is implemented and enabled, and [MDCR_EL2.E2PB](#) is 0b00, Secure EL1.
- If EL3 is implemented, and the owning Security state is Non-secure state, this field reads as one from:
 - Secure EL1.
 - If Secure EL2 is implemented, Secure EL2.
 - If EL2 is implemented and [MDCR_EL2.E2PB](#) is 0b00, Non-secure EL1.
 - If FEAT_RME is implemented, Realm EL1 and Realm EL2.
- If FEAT_RME is implemented, and the owning Security state is Realm state, this field reads as one from:
 - Non-secure EL1 and Non-secure EL2.
 - Secure EL1 and Secure EL2.
 - If [MDCR_EL2.E2PB](#) is 0b00, Realm EL1.
- If EL3 is not implemented, EL2 is implemented, and [MDCR_EL2.E2PB](#) is 0b00, this field reads as one from EL1.
- Otherwise, this field reads as zero.

Align, bits [3:0]

Defines the minimum alignment constraint for writes to [PMBPTR_EL1](#). Defined values are:

| Align | Meaning |
|--------|-------------|
| 0b0000 | Byte. |
| 0b0001 | Halfword. |
| 0b0010 | Word. |
| 0b0011 | Doubleword. |
| 0b0100 | 16 bytes. |
| 0b0101 | 32 bytes. |
| 0b0110 | 64 bytes. |
| 0b0111 | 128 bytes. |
| 0b1000 | 256 bytes. |
| 0b1001 | 512 bytes. |
| 0b1010 | 1KB. |
| 0b1011 | 2KB. |

All other values are reserved.

For more information, see 'Restrictions on the current write pointer'.

If this field is non-zero, then every record is a multiple of this size.

Access to this field is **RO**.

Accessing PMBIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMBIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1010 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMBIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return PMBIDR_EL1;
elsif PSTATE.EL == EL2 then
    return PMBIDR_EL1;
elsif PSTATE.EL == EL3 then
    return PMBIDR_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMBLIMITR_EL1, Profiling Buffer Limit Address Register

The PMBLIMITR_EL1 characteristics are:

Purpose

Defines the upper limit for the profiling buffer, and enables the profiling buffer

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMBLIMITR_EL1 are UNDEFINED.

Attributes

PMBLIMITR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|------|------|----|----|----|----|----|----|----|----|--|--|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | |
| | | | | | | | | | | | | | | | | LIMIT | | | | | | | | | | | | | | | | | | | | | |
| LIMIT | | | | | | | | | | | | | | | | RES0 | | | | | | PMFZ | RES0 | FM | E | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | |

LIMIT, bits [63:12]

Limit address. PMBLIMITR_EL1.LIMIT:Zeros(12) is the address of the first byte in memory after the last byte in the profiling buffer. If the smallest implemented translation granule is not 4KB, then bits[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value, $\text{Log}_2(\text{smallest implemented translation granule})$.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:6]

Reserved, RES0.

PMFZ, bit [5]

When FEAT_SPEv1p2 is implemented:

Freeze PMU on SPE event. Stop PMU event counters when [PMBSR_EL1.S](#) == 1.

| PMFZ | Meaning |
|------|--|
| 0b0 | Do not freeze PMU event counters on Statistical Profiling Buffer Management event. |
| 0b1 | Freeze PMU event counters on Statistical Profiling Buffer Management event. |

The PMU event counters affected by this control is controlled by [PMCR_EL0.FZS](#) and, if EL2 is implemented, [MDCR_EL2.HPMFZS](#). See the descriptions of these control bits for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [4:3]

Reserved, RES0.

FM, bits [2:1]

Fill mode.

| FM | Meaning | Applies when |
|------|--|----------------------------------|
| 0b00 | Fill mode. Stop collection and raise maintenance interrupt on buffer fill. | |
| 0b10 | Discard mode. All output is discarded. | When FEAT_SPEv1p2 is implemented |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Profiling Buffer enable

| E | Meaning |
|-----|---------------------------|
| 0b0 | All output is discarded. |
| 0b1 | Profiling buffer enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing PMBLIMITR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMBLIMITR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMBLIMITR_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            return NVMem[0x800];
        else
            return PMBLIMITR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMBLIMITR_EL1;
    elsif PSTATE.EL == EL3 then
        return PMBLIMITR_EL1;

```

MSR PMBLIMITR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMBLIMITR_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            NVMem[0x800] = X[t];
        else
            PMBLIMITR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMBLIMITR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMBLIMITR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMBPTR_EL1, Profiling Buffer Write Pointer Register

The PMBPTR_EL1 characteristics are:

Purpose

Defines the current write pointer for the profiling buffer.

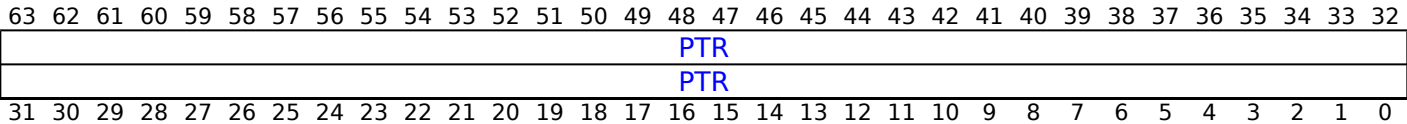
Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMBPTR_EL1 are UNDEFINED.

Attributes

PMBPTR_EL1 is a 64-bit register.

Field descriptions



PTR, bits [63:0]

Current write address. Defines the virtual address of the next entry to be written to the buffer.

The architecture places restrictions on the values software can write to the pointer. For more information see 'Restrictions on the current write pointer'.

Note

As a result, an implementation might treat some of bits[M:0], where M is defined by [PMBIDR_EL1](#).Align, as RES0.

- On a management interrupt, PMBPTR_EL1 is frozen.
- The reset behavior of this field is:
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMBPTR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMBPTR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMBPTR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            return NVMem[0x810];
        else
            return PMBPTR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMBPTR_EL1;
    elsif PSTATE.EL == EL3 then
        return PMBPTR_EL1;

```

MSR PMBPTR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMBPTR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            NVMem[0x810] = X[t];
        else
            PMBPTR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMBPTR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMBPTR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMBSR_EL1, Profiling Buffer Status/syndrome Register

The PMBSR_EL1 characteristics are:

Purpose

Provides syndrome information to software when the buffer is disabled because the management interrupt has been raised.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMBSR_EL1 are UNDEFINED.

Attributes

PMBSR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| EC | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| EC | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

EC, bits [31:26]

Exception class

Top-level description of the cause of the buffer management event

| EC | Meaning | MSS | Applies when |
|----------|--|---|------------------------------|
| 0b000000 | Other buffer management event. All buffer management events other than those described by other defined Exception class codes. | MSS encoding for other buffer management events | |
| 0b011110 | Granule Protection Check fault, other than GPF, on write to Profiling Buffer. | MSS encoding for Granule Protection Check fault | When FEAT_RME is implemented |
| 0b011111 | Buffer management event for an IMPLEMENTATION DEFINED reason. | MSS encoding for a buffer management event for an IMPLEMENTATION DEFINED reason | |
| 0b100100 | Stage 1 Data Abort on write to Profiling Buffer. | MSS encoding for stage 1 or stage 2 Data Aborts on write to buffer | |
| 0b100101 | Stage 2 Data Abort on write to Profiling Buffer. | MSS encoding for stage 1 or stage 2 Data Aborts on write to buffer | |

All other values are reserved. Reserved values might be defined in a future version of the architecture.

Writing a reserved value to this field will make the value of this field UNKNOWN. Values that are not supported act as reserved values when writing to this register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [25:20]

Reserved, RES0.

DL, bit [19]

Partial record lost.

Following a buffer management event other than an asynchronous External abort, indicates whether the last record written to the Profiling Buffer is complete.

| DL | Meaning |
|-----|--|
| 0b0 | PMBPTR_EL1 points to the first byte after the last complete record written to the Profiling Buffer. |
| 0b1 | Part of a record was lost because of a buffer management event or synchronous External abort. PMBPTR_EL1 might not point to the first byte after the last complete record written to the buffer, and so restarting collection might result in a data record stream that software cannot parse. All records prior to the last record have been written to the buffer. |

When the buffer management event was because of an asynchronous External abort, this bit is set to 1 and software must not assume that any valid data has been written to the Profiling Buffer.

This bit is RES0 if the PE never sets this bit as a result of a buffer management event caused by an asynchronous External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [18]

External abort.

| EA | Meaning |
|-----|--|
| 0b0 | An External abort has not been asserted. |
| 0b1 | An External abort has been asserted and detected by the Statistical Profiling Extension. |

This bit is RES0 if the PE never sets this bit as the result of an External abort.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [17]

Service

| S | Meaning |
|-----|--|
| 0b0 | PMBIRQ is not asserted. |
| 0b1 | PMBIRQ is asserted. All profiling data has either been written to the buffer or discarded. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COLL, bit [16]

Collision detected.

| COLL | Meaning |
|------|--|
| 0b0 | No collision events detected. |
| 0b1 | At least one collision event was recorded. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS, bits [15:0]

Management Event Specific Syndrome.

Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for stage 1 or stage 2 Data Aborts on write to buffer

| | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|---|---|---|---|-----|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | FSC | | | | | |

Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code

| FSC | Meaning | Applies when |
|------------|---|--|
| 0b000000 | Address size fault, level 0 of translation or translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000100 | Translation fault, level 0. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001000 | Access flag fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001100 | Permission fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010000 | Synchronous External abort, not on translation table walk or hardware update of translation table. | |
| 0b010001 | Asynchronous External abort. | |
| 0b010011 | Synchronous External abort on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented |
| 0b010100 | Synchronous External abort on translation table walk or hardware update of translation table, level 0. | |
| 0b010101 | Synchronous External abort on translation table walk or hardware update of translation table, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk or hardware update of translation table, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk or hardware update of translation table, level 3. | |
| 0b011011 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented and FEAT_RAS is not implemented |
| 0b100001 | Alignment fault. | |
| 0b100011 | Granule Protection Fault on translation table walk or hardware update of translation table, level -1. | When FEAT_RME is implemented and FEAT_LPA2 is implemented |
| 0b100100 | Granule Protection Fault on translation table walk or hardware update of translation table, level 0. | When FEAT_RME is implemented |
| 0b100101 | Granule Protection Fault on translation table walk or hardware update of translation table, level 1. | When FEAT_RME is implemented |

| | | |
|----------|--|---------------------------------|
| 0b100110 | Granule Protection Fault on translation table walk or hardware update of translation table, level 2. | When FEAT_RME is implemented |
| 0b100111 | Granule Protection Fault on translation table walk or hardware update of translation table, level 3. | When FEAT_RME is implemented |
| 0b101000 | Granule Protection Fault, not on translation table walk or hardware update of translation table. | When FEAT_RME is implemented |
| 0b101001 | Address size fault, level -1. | When FEAT_LPA2 is implemented |
| 0b101011 | Translation fault, level -1. | When FEAT_LPA2 is implemented |
| 0b110000 | TLB conflict abort. | |
| 0b110001 | Unsupported atomic hardware update fault. | When FEAT_HAFDBS is implemented |

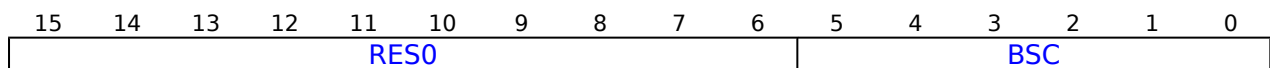
All other values are reserved.

It is IMPLEMENTATION DEFINED whether each of the Access Flag fault, asynchronous External abort and synchronous External abort, Alignment fault, and TLB Conflict abort values can be generated by the PE. For more information see 'Faults and Watchpoints'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for other buffer management events



Bits [15:6]

Reserved, RES0.

BSC, bits [5:0]

Buffer status code

| BSC | Meaning |
|----------|-------------------|
| 0b000000 | Buffer not filled |
| 0b000001 | Buffer filled |

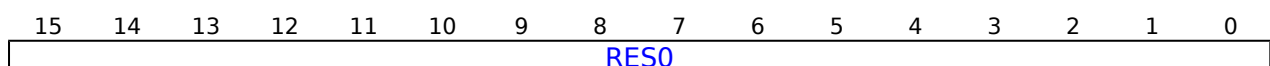
All other values are reserved. Reserved values might be defined in a future version of the architecture.

Writing a reserved value to this field will make the value of this field UNKNOWN. Values that are not supported act as reserved values when writing to this register.

The reset behavior of this field is:

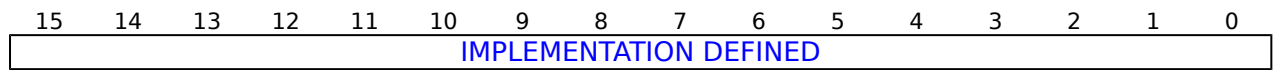
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for Granule Protection Check fault



Bits [15:0]

Reserved, RES0.

MSS encoding for a buffer management event for an IMPLEMENTATION DEFINED reason**IMPLEMENTATION DEFINED, bits [15:0]**

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMBSR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMBSR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMBSR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            return NVMem[0x820];
        else
            return PMBSR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMBSR_EL1;
    elsif PSTATE.EL == EL3 then
        return PMBSR_EL1;

```

MSR PMBSR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMBSR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2PB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            NVMem[0x820] = X[t];
        else
            PMBSR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMBSR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMBSR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCCFILTR_EL0, Performance Monitors Cycle Count Filter Register

The PMCCFILTR_EL0 characteristics are:

Purpose

Determines the modes in which the Cycle Counter, [PMCCNTR_EL0](#), increments.

Configuration

AArch64 System register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMCCFILTR\[31:0\]](#).

AArch64 System register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to External register [PMCCFILTR_EL0\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCCFILTR_EL0 are UNDEFINED.

Attributes

PMCCFILTR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|-----|-----|-----|----|------|----|----|-----|-----|-----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| P | U | NSK | NSU | NSH | M | RES0 | SH | T | RLK | RLU | RLH | RES0 | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

P, bit [31]

Privileged filtering bit. Controls counting in EL1.

If EL3 is implemented, then counting in Non-secure EL1 is further controlled by the PMCCFILTR_EL0.NSK bit.

If FEAT_RME is implemented, then counting in Realm EL1 is further controlled by the PMCCFILTR_EL0.RLK bit.

| P | Meaning |
|-----|-----------------------------|
| 0b0 | Count cycles in EL1. |
| 0b1 | Do not count cycles in EL1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

U, bit [30]

User filtering bit. Controls counting in EL0.

If EL3 is implemented, then counting in Non-secure EL0 is further controlled by the PMCCFILTR_EL0.NSU bit.

If FEAT_RME is implemented, then counting in Realm EL0 is further controlled by the PMCCFILTR_EL0.RLU bit.

| U | Meaning |
|-----|-----------------------------|
| 0b0 | Count cycles in EL0. |
| 0b1 | Do not count cycles in EL0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 (kernel) modes filtering bit. Controls counting in Non-secure EL1.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.P bit, cycles in Non-secure EL1 are counted.

Otherwise, cycles in Non-secure EL1 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When EL3 is implemented:

Non-secure EL0 (Unprivileged) filtering bit. Controls counting in Non-secure EL0.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.U bit, cycles in Non-secure EL0 are counted.

Otherwise, cycles in Non-secure EL0 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When EL2 is implemented:

EL2 (Hypervisor) filtering bit. Controls counting in EL2.

If Secure EL2 is implemented, and EL3 is implemented, counting in Secure EL2 is further controlled by the PMCCFILTR_EL0.SH bit.

If FEAT_RME is implemented, then counting in Realm EL2 is further controlled by the PMCCFILTR_EL0.RLH bit.

| NSH | Meaning |
|-----|-----------------------------|
| 0b0 | Do not count cycles in EL2. |
| 0b1 | Count cycles in EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

M, bit [26]

When EL3 is implemented:

Secure EL3 filtering bit.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.P bit, cycles in Secure EL3 are counted.

Otherwise, cycles in Secure EL3 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [25]

Reserved, RES0.

SH, bit [24]

When FEAT_SEL2 is implemented and EL3 is implemented:

Secure EL2 filtering.

If the value of this bit is not equal to the value of the PMCCFILTR_EL0.NSH bit, cycles in Secure EL2 are counted.

Otherwise, cycles in Secure EL2 are not counted.

If Secure EL2 is disabled, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

T, bit [23]

When FEAT_TME is implemented:

Non-transactional state filtering bit.

| T | Meaning |
|-----|---|
| 0b0 | This bit has no effect on filtering of cycles. |
| 0b1 | Do not count cycles in Non-transactional state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLK, bit [22]**When FEAT_RME is implemented:**

Realm EL1 (kernel) filtering bit. Controls counting in Realm EL1.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.P bit, cycles in Realm EL1 are counted.

Otherwise, cycles in Realm EL1 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLU, bit [21]**When FEAT_RME is implemented:**

Realm EL0 (unprivileged) filtering bit. Controls counting in Realm EL0.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.U bit, cycles in Realm EL0 are counted.

Otherwise, cycles in Realm EL0 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLH, bit [20]**When FEAT_RME is implemented:**

Realm EL2 filtering bit. Controls counting in Realm EL2.

If the value of this bit is not equal to the value of the PMCCFILTR_EL0.NSH bit, cycles in Realm EL2 are counted.

Otherwise, cycles in Realm EL2 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:0]

Reserved, RES0.

Accessing PMCCFILTR_EL0

PMCCFILTR_EL0 can also be accessed by using [PMXEVTYPER_EL0](#) with [PMSELR_EL0.SEL](#) set to 0b11111.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMCCFILTR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b1111 | 0b111 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCCFILTR_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return PMCCFILTR_EL0;
        elsif PSTATE.EL == EL1 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
                UNDEFINED;
            elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCCFILTR_EL0 ==
'1' then
                AArch64.SystemAccessTrap(EL2, 0x18);
            elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
                AArch64.SystemAccessTrap(EL2, 0x18);
            elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return PMCCFILTR_EL0;
        elsif PSTATE.EL == EL2 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
                UNDEFINED;
            elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return PMCCFILTR_EL0;
        elsif PSTATE.EL == EL3 then
            return PMCCFILTR_EL0;

```

MSR PMCCFILTR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1110 | 0b1111 | 0b111 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMCCFILTR_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCCFILTR_EL0 = X[t];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCCFILTR_EL0 ==
'1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCCFILTR_EL0 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCCFILTR_EL0 = X[t];
    elsif PSTATE.EL == EL3 then
        PMCCFILTR_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCCNTR_EL0, Performance Monitors Cycle Count Register

The PMCCNTR_EL0 characteristics are:

Purpose

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles. See 'Time as measured by the Performance Monitors cycle counter' for more information.

[PMCCFILTER_EL0](#) determines the modes and states in which the PMCCNTR_EL0 can increment.

Configuration

AArch64 System register PMCCNTR_EL0 bits [63:0] are architecturally mapped to AArch32 System register [PMCCNTR\[63:0\]](#).

AArch64 System register PMCCNTR_EL0 bits [63:0] are architecturally mapped to External register [PMCCNTR_EL0\[63:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCCNTR_EL0 are UNDEFINED.

All counters are subject to any changes in clock frequency, including clock stopping caused by the WFI and WFE instructions. This means that it is CONSTRAINED UNPREDICTABLE whether or not PMCCNTR_EL0 continues to increment when clocks are stopped by WFI and WFE instructions.

Attributes

PMCCNTR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | CCNT | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | CCNT | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

CCNT, bits [63:0]

Cycle count. Depending on the values of [PMCR_EL0](#).{LC,D}, this field increments in one of the following ways:

- Every processor clock cycle.
- Every 64th processor clock cycle.

Writing 1 to [PMCR_EL0](#).C sets this field to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCCNTR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMCCNTR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1101 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.<CR,EN> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMCCNTR_EL0;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCCNTR_EL0 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMCCNTR_EL0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMCCNTR_EL0;
elsif PSTATE.EL == EL3 then
    return PMCCNTR_EL0;

```

MSR PMCCNTR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1101 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCCNTR_EL0 = X[t];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCCNTR_EL0 == '1'
then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCCNTR_EL0 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCCNTR_EL0 = X[t];
    elsif PSTATE.EL == EL3 then
        PMCCNTR_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID0_EL0, Performance Monitors Common Event Identification register 0

The PMCEID0_EL0 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the ranges 0x0000 to 0x001F and 0x4000 to 0x401F.

For more information about the Common events and the use of the PMCEID<n>_EL0 registers see 'The PMU event number space and common events'.

Configuration

AArch64 System register PMCEID0_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMCEID0\[31:0\]](#).

AArch64 System register PMCEID0_EL0 bits [63:32] are architecturally mapped to AArch32 System register [PMCEID2\[31:0\]](#).

AArch64 System register PMCEID0_EL0 bits [31:0] are architecturally mapped to External register [PMCEID0\[31:0\]](#).

AArch64 System register PMCEID0_EL0 bits [63:32] are architecturally mapped to External register [PMCEID2\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCEID0_EL0 are UNDEFINED.

Attributes

PMCEID0_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | |
|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 |
| IDhi31 | IDhi30 | IDhi29 | IDhi28 | IDhi27 | IDhi26 | IDhi25 | IDhi24 | IDhi23 | IDhi22 | IDhi21 | IDhi20 | IDhi19 | IDhi18 | IDhi17 | IDhi16 | IDhi15 |
| ID31 | ID30 | ID29 | ID28 | ID27 | ID26 | ID25 | ID24 | ID23 | ID22 | ID21 | ID20 | ID19 | ID18 | ID17 | ID16 | ID15 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 |

IDhi<n>, bit [n+32], for n = 31 to 0

When FEAT_PMUv3p1 is implemented:

IDhi[n] corresponds to Common event (0x4000 + n).

For each bit:

| IDhi<n> | Meaning |
|---------|--|
| 0b0 | The Common event is not implemented, or not counted. |
| 0b1 | The Common event is implemented. |

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

Otherwise:

Reserved, RES0.

ID<n>, bit [n], for n = 31 to 0

ID[n] corresponds to Common event n.

For each bit:

| ID<n> | Meaning |
|-------|--|
| 0b0 | The Common event is not implemented, or not counted. |
| 0b1 | The Common event is implemented. |

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

Accessing PMCEID0_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMCEID0_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b110 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMCEID0_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCEIDn_EL0 == '1'
then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMCEID0_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMCEID0_EL0;
    elsif PSTATE.EL == EL3 then
        return PMCEID0_EL0;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID1_EL0, Performance Monitors Common Event Identification register 1

The PMCEID1_EL0 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the ranges 0x0020 to 0x003F and 0x4020 to 0x403F.

For more information about the Common events and the use of the PMCEID<n>_EL0 registers see 'The PMU event number space and common events'.

Configuration

AArch64 System register PMCEID1_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMCEID1\[31:0\]](#).

AArch64 System register PMCEID1_EL0 bits [63:32] are architecturally mapped to AArch32 System register [PMCEID3\[31:0\]](#).

AArch64 System register PMCEID1_EL0 bits [31:0] are architecturally mapped to External register [PMCEID1\[31:0\]](#).

AArch64 System register PMCEID1_EL0 bits [63:32] are architecturally mapped to External register [PMCEID3\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCEID1_EL0 are UNDEFINED.

Attributes

PMCEID1_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | |
|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 |
| IDhi31 | IDhi30 | IDhi29 | IDhi28 | IDhi27 | IDhi26 | IDhi25 | IDhi24 | IDhi23 | IDhi22 | IDhi21 | IDhi20 | IDhi19 | IDhi18 | IDhi17 | IDhi16 | IDhi15 |
| ID31 | ID30 | ID29 | ID28 | ID27 | ID26 | ID25 | ID24 | ID23 | ID22 | ID21 | ID20 | ID19 | ID18 | ID17 | ID16 | ID15 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 |

IDhi<n>, bit [n+32], for n = 31 to 0

When FEAT_PMUv3p1 is implemented:

IDhi[n] corresponds to Common event (0x4020 + n).

For each bit:

| IDhi<n> | Meaning |
|---------|--|
| 0b0 | The Common event is not implemented, or not counted. |
| 0b1 | The Common event is implemented. |

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

Otherwise:

Reserved, RES0.

ID<n>, bit [n], for n = 31 to 0

ID[n] corresponds to Common event (0x0020 + n).

For each bit:

| ID<n> | Meaning |
|-------|--|
| 0b0 | The Common event is not implemented, or not counted. |
| 0b1 | The Common event is implemented. |

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n>_EL0 registers of that earlier version of the PMU architecture.

Accessing PMCEID1_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMCEID1_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMCEID1_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCEIDn_EL0 == '1'
then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMCEID1_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMCEID1_EL0;
    elsif PSTATE.EL == EL3 then
        return PMCEID1_EL0;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCNTENCLR_EL0, Performance Monitors Count Enable Clear register

The PMCNTENCLR_EL0 characteristics are:

Purpose

Disables the Cycle Count Register, [PMCCNTR_EL0](#), and any implemented event counters [PMEVCNTR<n>](#). Reading this register shows which counters are enabled.

Configuration

AArch64 System register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMCNTENCLR\[31:0\]](#).

AArch64 System register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to External register [PMCNTENCLR_EL0\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCNTENCLR_EL0 are UNDEFINED.

Attributes

PMCNTENCLR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

C, bit [31]

[PMCCNTR_EL0](#) disable bit. Disables the cycle counter register. Possible values are:

| C | Meaning |
|-----|--|
| 0b0 | When read, means the cycle counter is disabled. When written, has no effect. |
| 0b1 | When read, means the cycle counter is enabled. When written, disables the cycle counter. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter disable bit for [PMEVCNTR<n>_EL0](#).

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in [MDCR_EL2](#).HPMN. Otherwise, N is the value in [PMCR_EL0](#).N.

| P<n> | Meaning |
|-------------------|---|
| 0b0 | When read, means that PMEVCNTR<n>_EL0 is disabled. When written, has no effect. |
| 0b1 | When read, means that PMEVCNTR<n>_EL0 is enabled. When written, disables PMEVCNTR<n>_EL0 . |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCNTENCLR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMCNTENCLR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b010 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCNTEN == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMCNTENCLR_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCNTEN == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMCNTENCLR_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMCNTENCLR_EL0;
    elsif PSTATE.EL == EL3 then
        return PMCNTENCLR_EL0;

```

MSR PMCNTENCLR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b010 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMCNTEN == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCNTENCLR_EL0 = X[t];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCNTENCLR_EL0 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCNTENCLR_EL0 = X[t];
    elsif PSTATE.EL == EL3 then
        PMCNTENCLR_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCNTENSET_EL0, Performance Monitors Count Enable Set register

The PMCNTENSET_EL0 characteristics are:

Purpose

Enables the Cycle Count Register, [PMCCNTR_EL0](#), and any implemented event counters [PMEVCNTR<n>](#). Reading this register shows which counters are enabled.

Configuration

AArch64 System register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMCNTENSET\[31:0\]](#).

AArch64 System register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to External register [PMCNTENSET_EL0\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCNTENSET_EL0 are UNDEFINED.

Attributes

PMCNTENSET_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

C, bit [31]

[PMCCNTR_EL0](#) enable bit. Enables the cycle counter register. Possible values are:

| C | Meaning |
|-----|---|
| 0b0 | When read, means the cycle counter is disabled. When written, has no effect. |
| 0b1 | When read, means the cycle counter is enabled. When written, enables the cycle counter. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter enable bit for [PMEVCNTR<n>_EL0](#).

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in [MDCR_EL2](#).HPMN. Otherwise, N is the value in [PMCR_EL0](#).N.

| P<n> | Meaning |
|-------------------|---|
| 0b0 | When read, means that PMEVCNTR<n>_EL0 is disabled. When written, has no effect. |
| 0b1 | When read, means that PMEVCNTR<n>_EL0 event counter is enabled. When written, enables PMEVCNTR<n>_EL0 . |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCNTENSET_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMCNTENSET_EL0

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMCNTEN == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMCNTENSET_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCNTEN == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMCNTENSET_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMCNTENSET_EL0;
    elsif PSTATE.EL == EL3 then
        return PMCNTENSET_EL0;

```

MSR PMCNTENSET_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMCNTEN == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCNTENSET_EL0 = X[t];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCNTENSET_EL0 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCNTENSET_EL0 = X[t];
    elsif PSTATE.EL == EL3 then
        PMCNTENSET_EL0 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCR_EL0, Performance Monitors Control Register

The PMCR_EL0 characteristics are:

Purpose

Provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

Configuration

AArch64 System register PMCR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMCR\[31:0\]](#).

AArch64 System register PMCR_EL0 bits [7:0] are architecturally mapped to External register [PMCR_EL0\[7:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCR_EL0 are UNDEFINED.

Attributes

PMCR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|------|-----|------|----|----|----|----|----|----|----|----|----|-----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | FZS |
| IMP | | | | | | | | IDCODE | | | | | | | | N | | | | RES0 | FZ0 | RES0 | LP | LC | DP | X | D | C | P | E | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:33]

Reserved, RES0.

FZS, bit [32]

When FEAT_SPEv1p2 is implemented:

Freeze-on-SPE event. Stop counters when [PMBLIMITR_EL1](#).{PMFZ,E} == {1,1} and [PMBSR_EL1](#).S == 1.

In the description of this field:

- If EL2 is implemented and is using AArch32, PMN is [HDCR](#).HPMN.
- If EL2 is implemented and is using AArch64, PMN is [MDCR_EL2](#).HPMN.
- If EL2 is not implemented, PMN is PMCR_EL0.N.

| FZS | Meaning |
|-----|---|
| 0b0 | Do not freeze on Statistical Profiling Buffer Management event. |
| 0b1 | Event counter PMEVCNTR<n>_EL0 does not count following a Statistical Profiling Buffer Management event if n is in the range of affected event counters. |

If PMN is not 0, this field affects the operation of event counters in the range [0 .. (PMN-1)].

This field does not affect the operation of other event counters and [PMCCNTR_EL0](#).

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset:
 - When AArch32 is supported, this field resets to 0.
 - When the implementation only supports execution in AArch64 state, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IMP, bits [31:24]

When FEAT_PMUv3p7 is not implemented:

Implementer code.

If this field is zero, then PMCR_EL0.IDCODE is RES0 and software must use [MIDR_EL1](#) to identify the PE.

Otherwise, this field and PMCR_EL0.IDCODE identify the PMU implementation to software. The implementer codes are allocated by Arm. A non-zero value has the same interpretation as [MIDR_EL1](#).Implementer.

Use of this field is deprecated.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

IDCODE, bits [23:16]

When PMCR_EL0.IMP != 0b00000000:

Identification code. Use of this field is deprecated.

Each implementer must maintain a list of identification codes that are specific to the implementer. A specific implementation is identified by the combination of the implementer code and the identification code.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

N, bits [15:11]

Indicates the number of event counters implemented. This value is in the range of 0b000000-0b11111. If the value is 0b000000, then only [PMCCNTR_EL0](#) is implemented. If the value is 0b11111, then [PMCCNTR_EL0](#) and 31 event counters are implemented.

When EL2 is implemented and enabled for the current Security state, reads of this field from EL1 and EL0 return the value of [MDCR_EL2](#).HPMN.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [10]

Reserved, RES0.

FZO, bit [9]**When FEAT_PMUv3p7 is implemented:**

Freeze-on-overflow. Stop event counters on overflow.

In the description of this field:

- If EL2 is implemented and is using AArch32, PMN is [HDCR](#).HPMN.
- If EL2 is implemented and is using AArch64, PMN is [MDCR_EL2](#).HPMN.
- If EL2 is not implemented, PMN is PMCR_EL0.N.

| FZO | Meaning |
|------------|---|
| 0b0 | Do not freeze on overflow. |
| 0b1 | Event counter PMEVCNTR<n>_EL0 does not count when PMOVSCLR_EL0 [(PMN-1):0] is nonzero and n is in the range of affected event counters. |

If PMN is not 0, this field affects the operation of event counters in the range [0 .. (PMN-1)].

This field does not affect the operation of other event counters and [PMCCNTR_EL0](#).

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [8]

Reserved, RES0.

LP, bit [7]**When FEAT_PMUv3p5 is implemented:**

Long event counter enable. Determines when unsigned overflow is recorded by an event counter overflow bit.

In the description of this field:

- If EL2 is implemented and is using AArch32, PMN is [HDCR](#).HPMN.
- If EL2 is implemented and is using AArch64, PMN is [MDCR_EL2](#).HPMN.
- If EL2 is not implemented, PMN is PMCR_EL0.N.

| LP | Meaning |
|-----------|--|
| 0b0 | Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0 [31:0]. |
| 0b1 | Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0 [63:0]. |

If PMN is not 0, this field affects the operation of event counters in the range [0 .. (PMN-1)].

This field does not affect the operation of other event counters and [PMCCNTR_EL0](#).

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LC, bit [6]**When AArch32 is supported:**

Long cycle counter enable. Determines when unsigned overflow is recorded by the cycle counter overflow bit.

| LC | Meaning |
|-----|--|
| 0b0 | Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0 [31:0]. |
| 0b1 | Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0 [63:0]. |

Arm deprecates use of [PMCR_EL0](#).LC = 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

DP, bit [5]**When EL3 is implemented or (FEAT_PMuV3p1 is implemented and EL2 is implemented):**

Disable cycle counter when event counting is prohibited.

| DP | Meaning |
|-----|---|
| 0b0 | Cycle counting by PMCCNTR_EL0 is not affected by this mechanism. |
| 0b1 | Cycle counting by PMCCNTR_EL0 is disabled in prohibited regions: <ul style="list-style-type: none"> If FEAT_PMuV3p1 is implemented, EL2 is implemented, and MDCR_EL2.HPMD is 1, then cycle counting by PMCCNTR_EL0 is disabled at EL2. If FEAT_PMuV3p7 is implemented, EL3 is implemented and using AArch64, and MDCR_EL3.MPMX is 1, then cycle counting by PMCCNTR_EL0 is disabled at EL3. If EL3 is implemented, MDCR_EL3.SPME is 0, and either FEAT_PMuV3p7 is not implemented or MDCR_EL3.MPMX is 0, then cycle counting by PMCCNTR_EL0 is disabled at EL3 and in Secure state. <p>If MDCR_EL2.HPMN is not 0, this is when event counting by event counters in the range [0..(MDCR_EL2.HPMN-1)] is prohibited.</p> |

For more information see 'Prohibiting event counting'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

X, bit [4]

When the implementation includes a PMU event export bus:

Enable export of events in an IMPLEMENTATION DEFINED PMU event export bus.

| X | Meaning |
|-----|-------------------------------------|
| 0b0 | Do not export events. |
| 0b1 | Export events where not prohibited. |

This field enables the exporting of events over an IMPLEMENTATION DEFINED PMU event export bus to another device, for example to an OPTIONAL PE trace unit.

No events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

D, bit [3]

When AArch32 is supported:

Clock divider.

| D | Meaning |
|-----|--|
| 0b0 | When enabled, PMCCNTR_EL0 counts every clock cycle. |
| 0b1 | When enabled, PMCCNTR_EL0 counts once every 64 clock cycles. |

If PMCR_EL0.LC == 1, this bit is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR_EL0.D = 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

C, bit [2]

Cycle counter reset. The effects of writing to this bit are:

| C | Meaning |
|-----|--|
| 0b0 | No action. |
| 0b1 | Reset PMCCNTR_EL0 to zero. |

Note

Resetting [PMCCNTR_EL0](#) does not change the cycle counter overflow bit. If FEAT_PMUv3p5 is implemented, the value of PMCR_EL0.LC is ignored, and bits [63:0] of the cycle counter are reset.

Access to this field is **WO/RAZ**.

P, bit [1]

Event counter reset.

In the description of this field:

- If EL2 is implemented and is using AArch32, PMN is [HDCR](#).HPMN.
- If EL2 is implemented and is using AArch64, PMN is [MDCR_EL2](#).HPMN.
- If EL2 is not implemented, PMN is PMCR_EL0.N.

| P | Meaning |
|-----|---|
| 0b0 | No action. |
| 0b1 | If n is in the range of affected event counters, resets each event counter PMEVCNTR<n> to zero. |

The effects of writing to this bit are:

- If EL2 is implemented and enabled in the current Security state, in EL0 and EL1, if PMN is not 0, a write of 1 to this bit resets event counters in the range [0 .. (PMN-1)].
- If EL2 is disabled in the current Security state, a write of 1 to this bit resets all the event counters.
- In EL2 and EL3, a write of 1 to this bit resets all the event counters.
- This field does not affect the operation of other event counters and [PMCCNTR_EL0](#).

Note

Resetting the event counters does not change the event counter overflow bits. If FEAT_PMUv3p5 is implemented, the values of [MDCR_EL2](#).HLP and PMCR_EL0.LP are ignored, and bits [63:0] of all affected event counters are reset.

Access to this field is **WO/RAZ**.

E, bit [0]

Enable.

If EL2 is implemented and is using AArch32, PMN is [HDCR](#).HPMN.

If EL2 is implemented and is using AArch64, PMN is [MDCR_EL2](#).HPMN.

If EL2 is not implemented, PMN is PMCR_EL0.N.

| E | Meaning |
|-----|--|
| 0b0 | PMCCNTR_EL0 is disabled and event counters PMEVCNTR<n>_EL0 , where n is in the range of affected event counters, are disabled. |
| 0b1 | PMCCNTR_EL0 and event counters PMEVCNTR<n>_EL0 , where n is in the range of affected event counters, are enabled by PMCNTENSET_EL0 . |

If PMN is not 0, this field affects the operation of event counters in the range [0 .. (PMN-1)].

This field does not affect the operation of other event counters.

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing PMCR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMCR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMCR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMCR_EL0;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMCR == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMCR_EL0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMCR_EL0;
elsif PSTATE.EL == EL3 then
    return PMCR_EL0;

```

MSR PMCR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMCR_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPMCR == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCR_EL0 = X[t];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCR_EL0 == '1'
then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPMCR == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCR_EL0 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMCR_EL0 = X[t];
    elsif PSTATE.EL == EL3 then
        PMCR_EL0 = X[t];

```

PMEVCNTR<n>_EL0, Performance Monitors Event Count Registers, n = 0 - 30

The PMEVCNTR<n>_EL0 characteristics are:

Purpose

Holds event counter n, which counts events, where n is 0 to 30.

Configuration

AArch64 System register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMEVCNTR<n>\[31:0\]](#).

AArch64 System register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to External register [PMEVCNTR<n>_EL0\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMEVCNTR<n>_EL0 are UNDEFINED.

Attributes

PMEVCNTR<n>_EL0 is a 64-bit register.

Field descriptions

When FEAT_PMUv3p5 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Event counter n | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Event counter n | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Event counter n | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

Bits [31:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMEVCNTR<n>_EL0

PMEVCNTR<n>_EL0 can also be accessed by using [PMXEVCNTR_EL0](#) with [PMSELR_EL0](#).SEL set to the value of <n>.

If FEAT_FGT is implemented and <n> is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of [PMEVCNTR<n>_EL0](#) is as follows:

- If <n> is an unimplemented event counter, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and <n> is greater than or equal to the number of accessible event counters, then reads and writes of [PMEVCNTR<n>_EL0](#) are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- Accesses to the register behave as if <n> is an UNKNOWN value less-than-or-equal-to the index of the highest accessible event counter.
- If EL2 is implemented and enabled in the current Security state, and <n> is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Note

In EL0, an access is permitted if it is enabled by [PMUSERENR_EL0](#).{ER,EN}.

If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, [MDCR_EL2](#).HPMN identifies the number of accessible event counters. Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see [MDCR_EL2](#).HPMN.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMEVCNTR<n>_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|--------|
| 0b11 | 0b011 | 0b1110 | 0b10:n[4:3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVCNTRn_EL0 ==
'1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)];
    elsif PSTATE.EL == EL3 then
        return PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)];

```

MSR PMEVCNTR<n>_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|--------|
| 0b11 | 0b011 | 0b1110 | 0b10:n[4:3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVCNTRn_EL0 ==
'1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
    elsif PSTATE.EL == EL3 then
        PMEVCNTR_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMEVTYPER<n>_EL0, Performance Monitors Event Type Registers, n = 0 - 30

The PMEVTYPER<n>_EL0 characteristics are:

Purpose

Configures event counter n, where n is 0 to 30.

Configuration

AArch64 System register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMEVTYPER<n>\[31:0\]](#).

AArch64 System register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to External register [PMEVTYPER<n>_EL0\[31:0\]](#).

AArch64 System register PMEVTYPER<n>_EL0 bits [63:32] are architecturally mapped to External register [PMEVFILTR<n>\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMEVTYPER<n>_EL0 are UNDEFINED.

Attributes

PMEVTYPER<n>_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|-----|------|-----|----|----|----|----|-----|-----|-----|------|----|----|----|-----------------|----|----|----|---------------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| TC | | | RES0 | | | | | | | | | | | | | | | | | | TH | | | | | | | | | | |
| P | U | NSK | NSU | NSH | M | MT | SH | T | RLK | RLU | RLH | RES0 | | | | evtCount[15:10] | | | | evtCount[9:0] | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

TC, bits [63:61]

When FEAT_PMUv3_TH is implemented:

Threshold Control. Defines the threshold function. In the description of this field, the value V is the value the event specified by PMEVTYPER<n>_EL0 would increment the counter by on a processor cycle if the threshold function is disabled. Comparisons treat V and PMEVTYPER<n>_EL0.TH as unsigned integer values.

| TC | Meaning |
|-------|--|
| 0b000 | Not-equal. The counter increments by V on each processor cycle when V is not equal to PMEVTYPER<n>_EL0.TH. If PMEVTYPER<n>_EL0.TH is zero, the threshold function is disabled. |
| 0b001 | Not-equal, count. The counter increments by 1 on each processor cycle when V is not equal to PMEVTYPER<n>_EL0.TH. |
| 0b010 | Equals. The counter increments by V on each processor cycle when V is equal to PMEVTYPER<n>_EL0.TH. |
| 0b011 | Equals, count. The counter increments by 1 on each processor cycle when V is equal to PMEVTYPER<n>_EL0.TH. |
| 0b100 | Greater-than-or-equal. The counter increments by V on each processor cycle when V is PMEVTYPER<n>_EL0.TH or more. |
| 0b101 | Greater-than-or-equal, count. The counter increments by 1 on each processor cycle when V is PMEVTYPER<n>_EL0.TH or more. |
| 0b110 | Less-than. The counter increments by V on each processor cycle when V is less than PMEVTYPER<n>_EL0.TH. |
| 0b111 | Less-than, count. The counter increments by 1 on each processor cycle when V is less than PMEVTYPER<n>_EL0.TH. |

The reset behavior of this field is:

- On a Warm reset:
 - When AArch32 is supported, this field resets to 0.
 - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [60:44]

Reserved, RES0.

TH, bits [43:32]

When FEAT_PMUv3_TH is implemented:

Threshold value. Provides the unsigned value for the threshold function defined by PMEVTYPER<n>_EL0.TC.

If PMEVTYPER<n>_EL0.TC is 0b000 and PMEVTYPER<n>_EL0.TH is zero, then the threshold function is disabled.

If [PMMIR_EL1](#).THWIDTH is less than 12, then bits PMEVTYPER<n>_EL0.TH[11:[PMMIR_EL1](#).THWIDTH] are RES0. This accounts for the behavior when writing a value greater-than-or-equal-to $2^{(\text{PMMIR_EL1.THWIDTH})}$.

The reset behavior of this field is:

- On a Warm reset:
 - When AArch32 is supported, this field resets to 0.
 - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

P, bit [31]

Privileged filtering bit. Controls counting in EL1.

If EL3 is implemented, then counting in Non-secure EL1 is further controlled by the PMEVTYPER<n>_EL0.NSK bit.

If FEAT_RME is implemented, then counting in Realm EL1 is further controlled by the PMEVTYPER<n>_EL0.RLK bit.

| P | Meaning |
|----------|-----------------------------|
| 0b0 | Count events in EL1. |
| 0b1 | Do not count events in EL1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

U, bit [30]

User filtering bit. Controls counting in EL0.

If EL3 is implemented, then counting in Non-secure EL0 is further controlled by the PMEVTYPER<n>_EL0.NSU bit.

If FEAT_RME is implemented, then counting in Realm EL0 is further controlled by the PMEVTYPER<n>_EL0.RLU bit.

| U | Meaning |
|----------|-----------------------------|
| 0b0 | Count events in EL0. |
| 0b1 | Do not count events in EL0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 (kernel) modes filtering bit. Controls counting in Non-secure EL1.

If the value of this bit is equal to the value of the PMEVTYPER<n>_EL0.P bit, events in Non-secure EL1 are counted.

Otherwise, events in Non-secure EL1 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When EL3 is implemented:

Non-secure EL0 (Unprivileged) filtering bit. Controls counting in Non-secure EL0.

If the value of this bit is equal to the value of the PMEVTYPER<n>_EL0.U bit, events in Non-secure EL0 are counted.

Otherwise, events in Non-secure EL0 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]**When EL2 is implemented:**

EL2 (Hypervisor) filtering bit. Controls counting in EL2.

If Secure EL2 is implemented, and EL3 is implemented, counting in Secure EL2 is further controlled by the PMEVTYPER<n>_EL0.SH bit.

If FEAT_RME is implemented, then counting in Realm EL2 is further controlled by the PMEVTYPER<n>_EL0.RLH bit.

| NSH | Meaning |
|-----|-----------------------------|
| 0b0 | Do not count events in EL2. |
| 0b1 | Count events in EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

M, bit [26]**When EL3 is implemented:**

EL3 filtering bit.

If the value of this bit is equal to the value of the PMEVTYPER<n>_EL0.P bit, events in EL3 are counted.

Otherwise, events in EL3 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MT, bit [25]**When FEAT_MTPMU is implemented or an IMPLEMENTATION DEFINED multi-threaded PMU extension is implemented:**

Multithreading.

| MT | Meaning |
|-----|--|
| 0b0 | Count events only on controlling PE. |
| 0b1 | Count events from any PE with the same affinity at level 1 and above as this PE. |

From Armv8.6, the IMPLEMENTATION DEFINED multi-threaded PMU extension is not permitted, meaning if FEAT_MTPMU is not implemented, this field is RES0. See [ID_AA64DFR0_EL1](#).MTPMU.

This field is ignored by the PE and treated as zero when FEAT_MTPMU is implemented and Disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SH, bit [24]**When FEAT_SEL2 is implemented and EL3 is implemented:**

Secure EL2 filtering.

If the value of this bit is not equal to the value of the PMEVTYPER<n>_EL0.NSH bit, events in Secure EL2 are counted.

Otherwise, events in Secure EL2 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

T, bit [23]**When FEAT_TME is implemented:**

Transactional state filtering bit. Controls counting in Transactional state.

| T | Meaning |
|----------|--|
| 0b0 | This bit has no effect on the filtering of events. |
| 0b1 | Do not count events in Transactional state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLK, bit [22]**When FEAT_RME is implemented:**

Realm EL1 (kernel) filtering bit. Controls counting in Realm EL1.

If the value of this bit is equal to the value of the PMEVTYPER<n>_EL0.P bit, events in Realm EL1 are counted.

Otherwise, events in Realm EL1 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLU, bit [21]**When FEAT_RME is implemented:**

Realm EL0 (unprivileged) filtering bit. Controls counting in Realm EL0.

If the value of this bit is equal to the value of the PMEVTYPER<n>_EL0.U bit, events in Realm EL0 are counted.

Otherwise, events in Realm EL0 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLH, bit [20]**When FEAT_RME is implemented:**

Realm EL2 filtering bit. Controls counting in Realm EL2.

If the value of this bit is not equal to the value of the PMEVTYPER<n>_EL0.NSH bit, events in Realm EL2 are counted.

Otherwise, events in Realm EL2 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:16]

Reserved, RES0.

evtCount[15:10], bits [15:10]**When FEAT_PMUv3p1 is implemented:**

Extension to evtCount[9:0]. For more information, see evtCount[9:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

evtCount[9:0], bits [9:0]

Event to count.

The event number of the event that is counted by event counter [PMEVCNTR<n>_EL0](#).

The ranges of event numbers allocated to each type of event are shown in 'Allocation of the PMU event number space'.

If FEAT_PMUv3p8 is implemented and PMEVTYPER<n>_EL0.evtCount is programmed to an event that is reserved or not supported by the PE, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.

Otherwise, if PMEVTYPER<n>_EL0.evtCount is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range 0x0000 to 0x003F, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.
- If FEAT_PMUv3p1 is implemented, for the range 0x4000 to 0x403F, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.
- For other values, it is UNPREDICTABLE what event, if any, is counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is UNKNOWN.

Note

UNPREDICTABLE means the event must not expose privileged information.

Arm recommends that for all values that represent reserved or unsupported events, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMEVTYPER<n>_EL0

PMEVTYPER<n>_EL0 can also be accessed by using [PMXEVTYPER_EL0](#) with [PMSELR_EL0.SEL](#) set to n.

If FEAT_FGT is implemented and <n> is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of [PMEVTYPER<n>_EL0](#) is as follows:

- If <n> is an unimplemented event counter, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and <n> is greater than or equal to the number of accessible event counters, then reads and writes of [PMEVTYPER<n>_EL0](#) are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- Accesses to the register behave as if <n> is an UNKNOWN value less-than-or-equal-to the index of the highest accessible event counter.
- If EL2 is implemented and enabled in the current Security state, and <n> is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Note

In EL0, an access is permitted if it is enabled by [PMUSERENR_EL0.EN](#).

If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, [MDCR_EL2.HPMN](#) identifies the number of accessible event counters. Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see [MDCR_EL2.HPMN](#).

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMEVTYPER<n>_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|--------|
| 0b11 | 0b011 | 0b1110 | 0b11:n[4:3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMEVTYPEPERn_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMEVTYPEPER_EL0[UInt(CRm<1:0>:op2<2:0>)];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVTYPEPERn_EL0 ==
'1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMEVTYPEPER_EL0[UInt(CRm<1:0>:op2<2:0>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMEVTYPEPER_EL0[UInt(CRm<1:0>:op2<2:0>)];
    elsif PSTATE.EL == EL3 then
        return PMEVTYPEPER_EL0[UInt(CRm<1:0>:op2<2:0>)];

```

MSR PMEVTYPEPER<n>_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|--------|
| 0b11 | 0b011 | 0b1110 | 0b11:n[4:3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMEVTYPEPERn_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMEVTYPER_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVTYPEPERn_EL0 ==
'1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMEVTYPER_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMEVTYPER_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];
    elsif PSTATE.EL == EL3 then
        PMEVTYPER_EL0[UInt(CRm<1:0>:op2<2:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMINTENCLR_EL1, Performance Monitors Interrupt Enable Clear register

The PMINTENCLR_EL1 characteristics are:

Purpose

Disables the generation of interrupt requests on overflows from the Cycle Count Register, [PMCCNTR_EL0](#), and the event counters [PMEVCNTR<n>_EL0](#). Reading the register shows which overflow interrupt requests are enabled.

Configuration

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [PMINTENCLR\[31:0\]](#).

AArch64 System register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to External register [PMINTENCLR_EL1\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMINTENCLR_EL1 are UNDEFINED.

Attributes

PMINTENCLR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

C, bit [31]

[PMCCNTR_EL0](#) overflow interrupt request disable bit. Possible values are:

| C | Meaning |
|-----|--|
| 0b0 | When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect. |
| 0b1 | When read, means the cycle counter overflow interrupt request is enabled. When written, disables the cycle count overflow interrupt request. |

P<n>, bit [n], for n = 30 to 0

Event counter overflow interrupt request disable bit for [PMEVCNTR<n>_EL0](#).

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1, N is the value in [MDCR_EL2](#).HPMN. Otherwise, N is the value in [PMCR_EL0](#).N.

| P<n> | Meaning |
|------|---|
| 0b0 | When read, means that the PMEVCNTR<n>_EL0 event counter interrupt request is disabled. When written, has no effect. |
| 0b1 | When read, means that the PMEVCNTR<n>_EL0 event counter interrupt request is enabled. When written, disables the PMEVCNTR<n>_EL0 interrupt request. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMINTENCLR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMINTENCLR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMINTENCLR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMINTENCLR_EL1;
elsif PSTATE.EL == EL3 then
    return PMINTENCLR_EL1;

```

MSR PMINTENCLR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMINTENCLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMINTENCLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMINTENCLR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMINTENSET_EL1, Performance Monitors Interrupt Enable Set register

The PMINTENSET_EL1 characteristics are:

Purpose

Enables the generation of interrupt requests on overflows from the Cycle Count Register, [PMCCNTR_EL0](#), and the event counters [PMEVCNTR<n>_EL0](#). Reading the register shows which overflow interrupt requests are enabled.

Configuration

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register [PMINTENSET\[31:0\]](#).

AArch64 System register PMINTENSET_EL1 bits [31:0] are architecturally mapped to External register [PMINTENSET_EL1\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMINTENSET_EL1 are UNDEFINED.

Attributes

PMINTENSET_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

C, bit [31]

[PMCCNTR_EL0](#) overflow interrupt request enable bit. Possible values are:

| C | Meaning |
|-----|---|
| 0b0 | When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect. |
| 0b1 | When read, means the cycle counter overflow interrupt request is enabled. When written, enables the cycle count overflow interrupt request. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter overflow interrupt request enable bit for [PMEVCNTR<n>_EL0](#).

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1, N is the value in [MDCR_EL2](#).HPMN. Otherwise, N is the value in [PMCR_EL0](#).N.

| P<n> | Meaning |
|------|--|
| 0b0 | When read, means that the PMEVCNTR<n>_EL0 event counter interrupt request is disabled. When written, has no effect. |
| 0b1 | When read, means that the PMEVCNTR<n>_EL0 event counter interrupt request is enabled. When written, enables the PMEVCNTR<n>_EL0 interrupt request. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMINTENSET_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMINTENSET_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMINTENSET_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMINTENSET_EL1;
elsif PSTATE.EL == EL3 then
    return PMINTENSET_EL1;

```

MSR PMINTENSET_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.PMINTEN == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMINTENSET_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMINTENSET_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMINTENSET_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMMIR_EL1, Performance Monitors Machine Identification Register

The PMMIR_EL1 characteristics are:

Purpose

Describes Performance Monitors parameters specific to the implementation to software.

Configuration

This register is present only when FEAT_PMuV3p4 is implemented. Otherwise, direct accesses to PMMIR_EL1 are UNDEFINED.

Attributes

PMMIR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|---------|----|----|----|-----------|----|----|----|-----------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | THWIDTH | | | | BUS_WIDTH | | | | BUS_SLOTS | | | | | | | | SLOTS | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

THWIDTH, bits [23:20]

[PMEVTYPER<n>_EL0](#).TH width. Indicates implementation of the FEAT_PMuV3_TH feature, and, if implemented, the size of the [PMEVTYPER<n>_EL0](#).TH field.

| THWIDTH | Meaning |
|---------|--|
| 0b0000 | FEAT_PMuV3_TH is not implemented. |
| 0b0001 | 1 bit. PMEVTYPER<n>_EL0 .TH[11:1] are RES0. |
| 0b0010 | 2 bits. PMEVTYPER<n>_EL0 .TH[11:2] are RES0. |
| 0b0011 | 3 bits. PMEVTYPER<n>_EL0 .TH[11:3] are RES0. |
| 0b0100 | 4 bits. PMEVTYPER<n>_EL0 .TH[11:4] are RES0. |
| 0b0101 | 5 bits. PMEVTYPER<n>_EL0 .TH[11:5] are RES0. |
| 0b0110 | 6 bits. PMEVTYPER<n>_EL0 .TH[11:6] are RES0. |
| 0b0111 | 7 bits. PMEVTYPER<n>_EL0 .TH[11:7] are RES0. |
| 0b1000 | 8 bits. PMEVTYPER<n>_EL0 .TH[11:8] are RES0. |
| 0b1001 | 9 bits. PMEVTYPER<n>_EL0 .TH[11:9] are RES0. |
| 0b1010 | 10 bits. PMEVTYPER<n>_EL0 .TH[11:10] are RES0. |
| 0b1011 | 11 bits. PMEVTYPER<n>_EL0 .TH[11] is RES0. |
| 0b1100 | 12 bits. |

All other values are reserved.

If FEAT_PMuV3_TH is not implemented, this field is zero.

Otherwise, the largest value that can be written to [PMEVTYPER<n>_EL0](#).TH is $2^{(\text{PMMIR_EL1.THWIDTH})}$ minus one.

Access to this field is **RO**.

BUS_WIDTH, bits [19:16]

Bus width. Indicates the number of bytes each BUS_ACCESS event relates to. Encoded as $\text{Log}_2(\text{number of bytes})$, plus one. Defined values are:

| BUS_WIDTH | Meaning |
|------------------|-----------------------------------|
| 0b0000 | The information is not available. |
| 0b0011 | Four bytes. |
| 0b0100 | 8 bytes. |
| 0b0101 | 16 bytes. |
| 0b0110 | 32 bytes. |
| 0b0111 | 64 bytes. |
| 0b1000 | 128 bytes. |
| 0b1001 | 256 bytes. |
| 0b1010 | 512 bytes. |
| 0b1011 | 1024 bytes. |
| 0b1100 | 2048 bytes. |

All other values are reserved.

Each transfer is up to this number of bytes. An access might be smaller than the bus width.

When this field is nonzero, each access counted by BUS_ACCESS is at most BUS_WIDTH bytes. An implementation might treat a wide bus as multiple narrower buses, such that a wide access on the bus increments the BUS_ACCESS counter by more than one.

Access to this field is **RO**.

BUS_SLOTS, bits [15:8]

Bus count. The largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle.

When this field is nonzero, the largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle is BUS_SLOTS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SLOTS, bits [7:0]

Operation width. The largest value by which the STALL_SLOT event might increment in a single cycle. If the STALL_SLOT event is not implemented, this field might read as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMMIR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMMIR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b000 | 0b1001 | 0b1110 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMMIR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMMIR_EL1;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMMIR_EL1;
elseif PSTATE.EL == EL3 then
    return PMMIR_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMOVSLR_EL0, Performance Monitors Overflow Flag Status Clear Register

The PMOVSLR_EL0 characteristics are:

Purpose

Contains the state of the overflow bit for the Cycle Count Register, [PMCCNTR_EL0](#), and each of the implemented event counters [PMEVCNTR<n>](#). Writing to this register clears these bits.

Configuration

AArch64 System register PMOVSLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMOVSRI\[31:0\]](#).

AArch64 System register PMOVSLR_EL0 bits [31:0] are architecturally mapped to External register [PMOVSLR_EL0\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMOVSLR_EL0 are UNDEFINED.

Attributes

PMOVSLR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

C, bit [31]

Cycle counter overflow clear bit.

| C | Meaning |
|-----|--|
| 0b0 | When read, means the cycle counter has not overflowed since this bit was last cleared. When written, has no effect. |
| 0b1 | When read, means the cycle counter has overflowed since this bit was last cleared. When written, clears the cycle counter overflow bit to 0. |

[PMCR_EL0](#).LC controls whether an overflow is detected from unsigned overflow of [PMCCNTR_EL0](#)[31:0] or unsigned overflow of [PMCCNTR_EL0](#)[63:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter overflow clear bit for [PMEVCNTR<n>_EL0](#).

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in [MDCR_EL2.HPMN](#). Otherwise, N is the value in [PMCR_EL0.N](#).

| P<n> | Meaning |
|-------------------|---|
| 0b0 | When read, means that PMEVCNTR<n>_EL0 has not overflowed since this bit was last cleared. When written, has no effect. |
| 0b1 | When read, means that PMEVCNTR<n>_EL0 has overflowed since this bit was last cleared. When written, clears the PMEVCNTR<n>_EL0 overflow bit to 0. |

If FEAT_PMUv3p5 is implemented, [MDCR_EL2.HLP](#) and [PMCR_EL0.LP](#) control whether an overflow is detected from unsigned overflow of [PMEVCNTR<n>_EL0](#)[31:0] or unsigned overflow of [PMEVCNTR<n>_EL0](#)[63:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMOVSLR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMOVSLR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b011 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMOVS == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMOVSLR_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMOVS == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMOVSLR_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMOVSLR_EL0;
    elsif PSTATE.EL == EL3 then
        return PMOVSLR_EL0;

```

MSR PMOVSLR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b011 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMOVS == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMOVSLR_EL0 = X[t];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMOVS == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMOVSLR_EL0 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMOVSLR_EL0 = X[t];
    elsif PSTATE.EL == EL3 then
        PMOVSLR_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMOVSSET_EL0, Performance Monitors Overflow Flag Status Set register

The PMOVSSET_EL0 characteristics are:

Purpose

Sets the state of the overflow bit for the Cycle Count Register, [PMCCNTR_EL0](#), and each of the implemented event counters [PMEVCNTR<n>](#).

Configuration

AArch64 System register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMOVSSETI\[31:0\]](#).

AArch64 System register PMOVSSET_EL0 bits [31:0] are architecturally mapped to External register [PMOVSSET_EL0\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMOVSSET_EL0 are UNDEFINED.

Attributes

PMOVSSET_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

C, bit [31]

Cycle counter overflow set bit.

| C | Meaning |
|-----|--|
| 0b0 | When read, means the cycle counter has not overflowed since this bit was last cleared. When written, has no effect. |
| 0b1 | When read, means the cycle counter has overflowed since this bit was last cleared. When written, sets the cycle counter overflow bit to 1. |

[PMCR_EL0](#).LC controls whether an overflow is detected from unsigned overflow of [PMCCNTR_EL0](#)[31:0] or unsigned overflow of [PMCCNTR_EL0](#)[63:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter overflow set bit for [PMEVCNTR<n>_EL0](#).

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in [MDCR_EL2.HPMN](#). Otherwise, N is the value in [PMCR_EL0.N](#).

| P<n> | Meaning |
|-------------------|---|
| 0b0 | When read, means that PMEVCNTR<n>_EL0 has not overflowed since this bit was last cleared. When written, has no effect. |
| 0b1 | When read, means that PMEVCNTR<n>_EL0 has overflowed since this bit was last cleared. When written, sets the PMEVCNTR<n>_EL0 overflow bit to 1. |

If FEAT_PMUv3p5 is implemented, [MDCR_EL2.HLP](#) and [PMCR_EL0.LP](#) control whether an overflow is detected from unsigned overflow of [PMEVCNTR<n>_EL0](#)[31:0] or unsigned overflow of [PMEVCNTR<n>_EL0](#)[63:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMOVSSET_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMOVSSET_EL0

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b011 | 0b1001 | 0b1110 | 0b011 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMOVS == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMOVSSET_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMOVS == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMOVSSET_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMOVSSET_EL0;
    elsif PSTATE.EL == EL3 then
        return PMOVSSET_EL0;

```

MSR PMOVSSET_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1110 | 0b011 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMOVS == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMOVSSET_EL0 = X[t];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMOVS == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMOVSSET_EL0 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMOVSSET_EL0 = X[t];
    elsif PSTATE.EL == EL3 then
        PMOVSSET_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSCR_EL1, Statistical Profiling Control Register (EL1)

The PMSCR_EL1 characteristics are:

Purpose

Provides EL1 controls for Statistical Profiling.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSCR_EL1 are UNDEFINED.

Attributes

PMSCR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|-------|-------|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | PCT | TS | PA | CX | RES0 | E1SPE | E0SPE | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:8]

Reserved, RES0.

PCT, bits [7:6]

When EL2 is implemented:

Physical Timestamp. If timestamp sampling is enabled and the Profiling Buffer is owned by EL1, requests which timestamp counter value is collected.

If FEAT_ECV is implemented, this is a two-bit field as shown. Otherwise, bit[7] is RES0.

| PCT | Meaning | Applies when |
|------|--|------------------------------|
| 0b00 | Virtual timestamp. The collected timestamp is the physical counter minus the value of CNTVOFF_EL2 . | |
| 0b01 | Physical timestamp. The collected timestamp is the physical counter. | |
| 0b11 | Guest physical timestamp. The collected timestamp is the physical counter minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2 : <ul style="list-style-type: none"> SCR_EL3.ECVEn == 0. CNTHCTL_EL2.ECV == 0. | When FEAT_ECV is implemented |

If EL2 is enabled in the current Security state, then the value of [PMSCR_EL2](#).PCT might override or modify the meaning of this field.

This field is ignored by the PE when the Profiling Buffer owning Exception level is EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Physical Timestamp. Reserved. This field reads as 0b01 and ignores writes. Software should treat this field as UNK/SBZP.

When EL2 is not implemented, the Effective values of [CNTVOFF_EL2](#) and [CNTPOFF_EL2](#) are zero, meaning the virtual counter and physical counter have the same value.

TS, bit [5]

Timestamp enable.

| TS | Meaning |
|-----|------------------------------|
| 0b0 | Timestamp sampling disabled. |
| 0b1 | Timestamp sampling enabled. |

This bit is ignored by the PE if EL2 is implemented and the Profiling Buffer is owned by EL2. For more information, see 'Controlling the data that is collected'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PA, bit [4]

Physical Address sample enable.

| PA | Meaning |
|-----|---------------------------------------|
| 0b0 | Physical addresses are not collected. |
| 0b1 | Physical addresses are collected. |

If EL2 is implemented:

- If the Profiling Buffer is owned by EL1, this bit is combined with [PMSCR_EL2.PA](#) to determine which address is collected. For more information, see 'Controlling the data that is collected'.
- If the Profiling Buffer is owned by EL2, this bit is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CX, bit [3]

[CONTEXTIDR_EL1](#) sample enable.

| CX | Meaning |
|-----|--|
| 0b0 | CONTEXTIDR_EL1 is not collected. |
| 0b1 | CONTEXTIDR_EL1 is collected. |

If EL2 is implemented and enabled in the current Security state when an operation is sampled:

- If the PE is at EL2, this bit is ignored by the PE.
- If [HCR_EL2.TGE](#) == 1, this bit is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E1SPE, bit [1]

EL1 Statistical Profiling Enable.

| E1SPE | Meaning |
|--------------|---------------------------|
| 0b0 | Sampling disabled at EL1. |
| 0b1 | Sampling enabled at EL1. |

If EL2 is implemented and enabled in the current Security state, this bit is ignored by the PE when [HCR_EL2.TGE](#) == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E0SPE, bit [0]

EL0 Statistical Profiling Enable. Controls sampling at EL0 when [HCR_EL2.TGE](#) == 0 or if EL2 is disabled or not implemented.

| E0SPE | Meaning |
|--------------|---------------------------|
| 0b0 | Sampling disabled at EL0. |
| 0b1 | Sampling enabled at EL0. |

If EL2 is implemented and enabled in the current Security state, this bit is ignored by the PE when [HCR_EL2.TGE](#) == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMSCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            return NVMem[0x828];
        else
            return PMSCR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            return PMSCR_EL2;
        else
            return PMSCR_EL1;
    elsif PSTATE.EL == EL3 then
        return PMSCR_EL1;

```

MSR PMSCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            NVMem[0x828] = X[t];
        else
            PMSCR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            PMSCR_EL2 = X[t];
        else
            PMSCR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMSCR_EL1 = X[t];

```

MRS <Xt>, PMSCR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1001 | 0b1001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x828];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMSCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return PMSCR_EL1;
    else
        UNDEFINED;

```

MSR PMSCR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1001 | 0b1001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x828] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSCR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        PMSCR_EL1 = X[t];
    else
        UNDEFINED;

```


PMSCR_EL2, Statistical Profiling Control Register (EL2)

The PMSCR_EL2 characteristics are:

Purpose

Provides EL2 controls for Statistical Profiling.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSCR_EL2 are UNDEFINED.

Attributes

PMSCR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|----|----|----|----|----|----|-----|----|----|----|----|------|-------|--------|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | PCT | | TS | PA | CX | RES0 | E2SPE | E0HSPE | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:8]

Reserved, RES0.

PCT, bits [7:6]

Physical Timestamp. If timestamp sampling is enabled, determines which counter is collected. The behavior depends on the Profiling Buffer owning Exception level.

If FEAT_ECV is implemented, this is a two-bit field as shown. Otherwise, bit[7] is RES0.

| PCT | Meaning | Applies when |
|------|--|------------------------------|
| 0b00 | <p>Virtual timestamp. The collected timestamp is the physical counter minus a virtual offset. If any of the following are true, the virtual offset is zero, otherwise the virtual offset is the value of CNTVOFF_EL2:</p> <ul style="list-style-type: none"> The sampled operation executed at EL2 and HCR_EL2.E2H == 1. The sampled operation executed at EL0 and HCR_EL2.{E2H,TGE} == {1,1}. <hr/> <p>Note If the Profiling Buffer owning Exception level is EL1, the virtual offset is always CNTVOFF_EL2.</p> | |
| 0b01 | <p>If the Profiling Buffer owning Exception level is EL1, then the timestamp value is selected by PMSCR_EL1.PCT. Otherwise, physical timestamp. The collected timestamp is the physical counter.</p> | |
| 0b11 | <p>If the Profiling Buffer owning Exception level is EL1 and PMSCR_EL1.PCT == 0b00, then guest virtual timestamp. The collected timestamp is the physical counter minus the value of CNTVOFF_EL2. Otherwise, guest physical timestamp. The collected timestamp is the physical counter minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2:</p> <ul style="list-style-type: none"> SCR_EL3.ECVEn == 0. CNTHCTL_EL2.ECV == 0. | When FEAT_ECV is implemented |

All other values are reserved.

If EL2 is not implemented or EL2 is disabled in the current Security state, then the Effective value of this field is 0b01, other than for a direct read of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TS, bit [5]

Timestamp Enable.

| TS | Meaning |
|-----|------------------------------|
| 0b0 | Timestamp sampling disabled. |
| 0b1 | Timestamp sampling enabled. |

This bit is ignored by the PE when any of the following are true:

- The Profiling Buffer owning Exception level is EL1.
- In Secure state, and either FEAT_SEL2 is not implemented or Secure EL2 is disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PA, bit [4]

Physical Address Sample Enable.

| PA | Meaning |
|-----|---------------------------------------|
| 0b0 | Physical addresses are not collected. |
| 0b1 | Physical addresses are collected. |

If the Profiling Buffer owning Exception level is EL1, and EL2 is enabled in the current Security state, this bit is combined with [PMSCR_EL1.PA](#) to determine which address is collected.

If EL2 is not implemented or EL2 is disabled in the current Security state, the PE ignores the value of this bit and behaves as if this bit is set to 1, other than for a direct read of the register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CX, bit [3]

[CONTEXTIDR_EL2](#) Sample Enable.

| CX | Meaning |
|-----|--|
| 0b0 | CONTEXTIDR_EL2 is not collected. |
| 0b1 | CONTEXTIDR_EL2 is collected. |

If EL2 is not implemented or EL2 is disabled in the current Security state, the PE ignores the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RES0.

E2SPE, bit [1]

EL2 Statistical Profiling Enable.

| E2SPE | Meaning |
|-------|---------------------------|
| 0b0 | Sampling disabled at EL2. |
| 0b1 | Sampling enabled at EL2. |

This bit is RES0 if [MDCR_EL2.E2PB](#) != 0b00.

If EL2 is disabled in the current Security state, this bit is ignored by the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EOHSPE, bit [0]

EL0 Statistical Profiling Enable.

| EOHSPE | Meaning |
|--------|---------------------------|
| 0b0 | Sampling disabled at EL0. |
| 0b1 | Sampling enabled at EL0. |

If [MDCR_EL2.E2PB](#) != 0b00, this bit is RES0.

If EL2 is implemented and enabled in the current Security state, this bit is ignored by the PE when [HCR_EL2.TGE](#) == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMSCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1001 | 0b1001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSCR_EL2;
elsif PSTATE.EL == EL3 then
    return PMSCR_EL2;

```

MSR PMSCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1001 | 0b1001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    PMSCR_EL2 = X[t];

```

MRS <Xt>, PMSCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            return NVMem[0x828];
        else
            return PMSCR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            return PMSCR_EL2;
        else
            return PMSCR_EL1;
    elsif PSTATE.EL == EL3 then
        return PMSCR_EL1;

```

MSR PMSCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            NVMem[0x828] = X[t];
        else
            PMSCR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            PMSCR_EL2 = X[t];
        else
            PMSCR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMSCR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSELR_EL0, Performance Monitors Event Counter Selection Register

The PMSELR_EL0 characteristics are:

Purpose

Selects the current event counter [PMEVCNTR<n>_EL0](#) or the cycle counter, CCNT.

PMSELR_EL0 is used in conjunction with [PMXEVTYPER_EL0](#) to determine the event that increments a selected event counter, and the modes and states in which the selected counter increments.

It is also used in conjunction with [PMXVCNTR_EL0](#), to determine the value of a selected event counter.

Configuration

AArch64 System register PMSELR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMSELR\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMSELR_EL0 are UNDEFINED.

Attributes

PMSELR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | SEL | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:5]

Reserved, RES0.

SEL, bits [4:0]

Selects event counter, [PMEVCNTR<n>_EL0](#), where n is the value held in this field. This value identifies which event counter is accessed when a subsequent access to [PMXEVTYPER_EL0](#) or [PMXVCNTR_EL0](#) occurs.

This field can take any value from 0 (0b000000) to (PMCR.N)-1, or 31 (0b111111).

When PMSELR_EL0.SEL is 0b111111, it selects the cycle counter and:

- A read of the [PMXEVTYPER_EL0](#) returns the value of [PMCCFILTR_EL0](#).
- A write of the [PMXEVTYPER_EL0](#) writes to [PMCCFILTR_EL0](#).
- A read or write of [PMXVCNTR_EL0](#) has CONSTRAINED UNPREDICTABLE effects. For more information, see [PMXVCNTR_EL0](#).

For more information about the results of accesses to the event counters, see [PMXEVTYPER_EL0](#) and [PMXVCNTR_EL0](#).

For more information about the number of counters accessible at each Exception level, see [MDCR_EL2](#).HPMN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSELR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMSELR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b101 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMSELR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSELR_EL0;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSELR_EL0 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSELR_EL0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSELR_EL0;
elsif PSTATE.EL == EL3 then
    return PMSELR_EL0;

```

MSR PMSELR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b101 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMSELR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSELR_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSELR_EL0 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSELR_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSELR_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    PMSELR_EL0 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSEVFR_EL1, Sampling Event Filter Register

The PMSEVFR_EL1 characteristics are:

Purpose

Controls sample filtering by events. The overall filter is the logical AND of these filters. For example, if E[3] and E[5] are both set to 1, only samples that have both event 3 (Level 1 unified or data cache refill) and event 5 set (TLB walk) are recorded.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSEVFR_EL1 are UNDEFINED.

Attributes

PMSEVFR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|-------|--------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | |
| E[63] | E[62] | E[61] | E[60] | E[59] | E[58] | E[57] | E[56] | E[55] | E[54] | E[53] | E[52] | E[51] | E[50] | E[49] | E[48] | | | | | | |
| E[31] | E[30] | E[29] | E[28] | E[27] | E[26] | E[25] | E[24] | RAZ/WI | | | | | E[18] | E[17] | E[16] | E[15] | E[14] | E[13] | E[12] | E[11] | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | |

E[<x>], bit [x], for x = 63 to 48, 31 to 24, 15 to 12

E[<x>] is the event filter for event <x>. If event <x> is not implemented, or filtering on event <x> is not supported, the corresponding bit is RAZ/WI.

| E[<x>] | Meaning |
|--------|---|
| 0b0 | Event <x> is ignored. |
| 0b1 | Do not record samples that have event <x> == 0. |

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, if the corresponding bits of PMSEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This field is ignored by the PE when [PMSECR_EL1.FE](#) == 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [47:32]

Reserved, RAZ/WI.

Bits [23:19]

Reserved, RAZ/WI.

E[18], bit [18]

When FEAT_SPEv1p1 is implemented and FEAT_SVE is implemented:

Empty predicate.

| E[18] | Meaning |
|--------------|---|
| 0b0 | Empty predicate event is ignored. |
| 0b1 | Do not record samples that have the Empty predicate event == 0. |

This bit is ignored by the PE when [PMSFCR_EL1](#).FE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[17], bit [17]

When FEAT_SPEv1p1 is implemented and FEAT_SVE is implemented:

Partial predicate.

| E[17] | Meaning |
|--------------|---|
| 0b0 | Partial predicate event is ignored. |
| 0b1 | Do not record samples that have the Partial predicate event == 0. |

This bit is ignored by the PE when [PMSFCR_EL1](#).FE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[16], bit [16]

When FEAT_TME is implemented:

Transactional

| E[16] | Meaning |
|--------------|---|
| 0b0 | Transactional event is ignored. |
| 0b1 | Do not record samples that have the Transactional event == 0. |

This bit is ignored by the PE when [PMSFCR_EL1](#).FE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[11], bit [11]

When FEAT_SPEv1p1 is implemented:

Alignment.

| E[11] | Meaning |
|--------------|---|
| 0b0 | Alignment event is ignored. |
| 0b1 | Do not record samples that have the Alignment event == 0. |

This bit is ignored by the PE when [PMSFCR_EL1](#).FE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bits [10:8]

Reserved, RAZ/WI.

E[7], bit [7]

Mispredicted.

| E[7] | Meaning |
|-------------|--|
| 0b0 | Mispredicted event is ignored. |
| 0b1 | Do not record samples that have the Mispredicted event == 0. |

This bit is ignored by the PE when [PMSFCR_EL1](#).FE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[6], bit [6]

When FEAT_SPEv1p2 is implemented:

Not taken.

| E[6] | Meaning |
|-------------|---|
| 0b0 | Not taken event is ignored. |
| 0b1 | Do not record samples that have the Not taken event == 0. |

This field is ignored by the PE when [PMSFCR_EL1](#).FE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[5], bit [5]

TLB walk.

| E[5] | Meaning |
|-------------|--|
| 0b0 | TLB walk event is ignored. |
| 0b1 | Do not record samples that have the TLB walk event == 0. |

This bit is ignored by the PE when [PMSFCR_EL1](#).FE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [4]

Reserved, RAZ/WI.

E[3], bit [3]

Level 1 data or unified cache refill.

| E[3] | Meaning |
|-------------|--|
| 0b0 | Level 1 data or unified cache refill event is ignored. |
| 0b1 | Do not record samples that have the Level 1 data or unified cache refill event == 0. |

This bit is ignored by the PE when [PMSFCR_EL1](#).FE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RAZ/WI.

E[1], bit [1]

When the PE supports sampling of speculative instructions:

Architecturally executed.

When the PE supports sampling of speculative instructions:

| E[1] | Meaning |
|-------------|--|
| 0b0 | Architecturally executed event is ignored. |
| 0b1 | Do not record samples that have the Architecturally executed event == 0. |

This bit is ignored by the PE when [PMSFCR_EL1](#).FE == 0.

If the PE does not support the sampling of speculative instructions, or always discards the sample record for speculative instructions, this bit reads as an UNKNOWN value and the PE ignores its value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, UNKNOWN.

Bit [0]

Reserved, RAZ/WI.

Accessing PMSEVFR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMSEVFR_EL1

| | | | | |
|------------|------------|------------|------------|------------|
| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b101 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSEVFR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            return NVMem[0x830];
        else
            return PMSEVFR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMSEVFR_EL1;
    elsif PSTATE.EL == EL3 then
        return PMSEVFR_EL1;

```

MSR PMSEVFR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSEVFR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            NVMem[0x830] = X[t];
        else
            PMSEVFR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSEVFR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMSEVFR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSFCR_EL1, Sampling Filter Control Register

The PMSFCR_EL1 characteristics are:

Purpose

Controls sample filtering. The filter is the logical AND of the FL, FT and FE bits. For example, if FE == 1 and FT == 1 only samples including the selected operation types and the selected events will be recorded

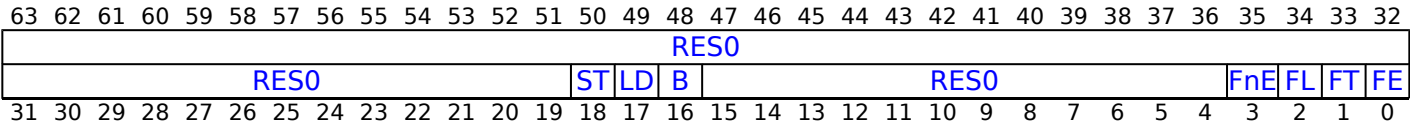
Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSFCR_EL1 are UNDEFINED.

Attributes

PMSFCR_EL1 is a 64-bit register.

Field descriptions



Bits [63:19]

Reserved, RES0.

ST, bit [18]

Store filter enable

| ST | Meaning |
|-----|--|
| 0b0 | Do not record store operations |
| 0b1 | Record all store operations, including vector stores and all atomic operations |

This bit is ignored by the PE when PMSFCR_EL1.FT == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LD, bit [17]

Load filter enable

| LD | Meaning |
|-----|---|
| 0b0 | Do not record load operations |
| 0b1 | Record all load operations, including vector loads and atomic operations that return data |

This bit is ignored by the PE when PMSFCR_EL1.FT == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

B, bit [16]

Branch filter enable

| B | Meaning |
|-----|--|
| 0b0 | Do not record branch and exception return operations |
| 0b1 | Record all branch and exception return operations |

This bit is ignored by the PE when PMSFCR_EL1.FT == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:4]

Reserved, RES0.

FnE, bit [3]

When FEAT_SPEv1p2 is implemented:

Filter by event, inverted.

| FnE | Meaning |
|-----|---|
| 0b0 | Inverted event filtering disabled. |
| 0b1 | Inverted event filtering enabled. Samples including the events selected by PMSNEVFR_EL1 will not be recorded. |

If any of the following are true, it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FnE == 0:

- PMSFCR_EL1.FnE == 1 and [PMSNEVFR_EL1](#) is zero.
- PMSFCR_EL1.FnE == 1, PMSFCR_EL1.FE == 1, and there exists a value x such that [PMSEVFR_EL1.E\[x\]](#) == 1 and [PMSNEVFR_EL1.E\[x\]](#) == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FL, bit [2]

Filter by latency

| FL | Meaning |
|-----|--|
| 0b0 | Latency filtering disabled |
| 0b1 | Latency filtering enabled. Samples with a total latency less than PMSLATFR_EL1.MINLAT will not be recorded |

If this field is set to 1 and PMSLATFR_EL1.MINLAT is set to zero, it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FL is set to 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FT, bit [1]

Filter by operation type. The filter is the logical OR of the ST, LD and B bits. For example, if LD and ST are both set, both load and store operations are recorded

| FT | Meaning |
|-----|--|
| 0b0 | Type filtering disabled |
| 0b1 | Type filtering enabled. Samples not one of the selected operation types will not be recorded |

If this field is set to 1 and the PMSFCR_EL1.{ST, LD, B} bits are all set to zero, it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FT is set to 0

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FE, bit [0]

Filter by event.

| FE | Meaning |
|-----|---|
| 0b0 | Event filtering disabled. |
| 0b1 | Event filtering enabled. Samples not including the events selected by PMSEVFR_EL1 will not be recorded. |

If any of the following are true, it is CONSTRAINED UNPREDICTABLE whether no samples are recorded or the PE behaves as if PMSFCR_EL1.FE == 0:

- PMSFCR_EL1.FE == 1 and [PMSEVFR_EL1](#) is zero.
- FEAT_SPEv1p2 is implemented, PMSFCR_EL1.FnE == 1, PMSFCR_EL1.FE == 1, and there exists a value x such that [PMSEVFR_EL1](#).E[x] == 1 and [PMSNEVER_EL1](#).E[x] == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSFCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMSFCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSFCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMSFCR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMSFCR_EL1;
elsif PSTATE.EL == EL3 then
    return PMSFCR_EL1;

```

MSR PMSFCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSFCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSFCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSFCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    PMSFCR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSICR_EL1, Sampling Interval Counter Register

The PMSICR_EL1 characteristics are:

Purpose

Software must write zero to PMSICR_EL1 before enabling sample profiling for a sampling session. Software must then treat PMSICR_EL1 as an opaque, 64-bit, read/write register used for context switches only.

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSICR_EL1 are UNDEFINED.

The value of PMSICR_EL1 does not change whilst profiling is disabled.

Attributes

PMSICR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ECOUNT | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | | | | | |
| COUNT | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ECOUNT, bits [63:56]

When PMSIDR_EL1.ERnd == 1:

Secondary sample interval counter.

This field provides the secondary counter used after the primary counter reaches zero. Whilst the secondary counter is nonzero and profiling is enabled, the secondary counter decrements by 1 for each member of the sample population. The primary counter also continues to decrement since it is also nonzero. When the secondary counter reaches zero, a member of the sampling population is selected for sampling.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [55:32]

Reserved, RES0.

COUNT, bits [31:0]

Primary sample interval counter

Provides the primary counter used for sampling.

The primary counter is reloaded when the value of this register is zero and the PE moves from a state or Exception level where profiling is disabled to a state or Exception level where profiling is enabled

Whilst the primary counter is nonzero and sampling is enabled, the primary counter decrements by 1 for each member of the sample population

When the counter reaches zero, the behavior depends on the values of PMSIDR_EL1.ERnd and PMSIRR_EL1.RND

- If [PMSIRR_EL1](#).RND == 0 or PMSIDR_EL1.ERnd == 0:
 - A member of the sampling population is selected for sampling
 - The primary counter is reloaded
- If [PMSIRR_EL1](#).RND == 1 and [PMSIDR_EL1](#).ERnd == 1:
 - The secondary counter is set to a random or pseudorandom value in the range 0x00 to 0xFF
 - The primary counter is reloaded

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSICR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMSICR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSICR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            return NVMem[0x838];
        else
            return PMSICR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMSICR_EL1;
    elsif PSTATE.EL == EL3 then
        return PMSICR_EL1;

```

MSR PMSICR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSICR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            NVMem[0x838] = X[t];
        else
            PMSICR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSICR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMSICR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSIDR_EL1, Sampling Profiling ID Register

The PMSIDR_EL1 characteristics are:

Purpose

Describes the Statistical Profiling implementation to software

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSIDR_EL1 are UNDEFINED.

Attributes

PMSIDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-----|--------|----|----|----|-----------|----|----|----|---------|----|----|----|----------|----|----|----|------|-----|------|-----|----------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | PBT | Format | | | | CountSize | | | | MaxSize | | | | Interval | | | | RES0 | FnE | ERnd | LDS | ArchInst | FL | FT | FE |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:25]

Reserved, RES0.

PBT, bit [24]

Previous branch target Address packet. Defined values are:

| PBT | Meaning |
|-----|--|
| 0b0 | Previous branch target Address packet not supported. |
| 0b1 | Previous branch target Address packet support implemented. |

FEAT_SPEv1p2 adds the OPTIONAL functionality identified by the value 1.

Format, bits [23:20]

From Armv8.7:

Defines the format of the sample records. Defined values are:

| Format | Meaning |
|--------|-----------|
| 0b0000 | Format 0. |

All other values are reserved.

Otherwise:

Reserved, RAZ.

CountSize, bits [19:16]

Defines the size of the counters.

| CountSize | Meaning | Applies when |
|-----------|-----------------------------|--------------|
| 0b0010 | 12-bit saturating counters. | |
| 0b0011 | 16-bit saturating counters. | From Armv8.8 |

All other values are reserved.

Access to this field is **RO**.

MaxSize, bits [15:12]

Defines the largest size for a single record, rounded up to a power-of-two. If this is the same as the minimum alignment ([PMBIDR_EL1.Align](#)), then each record is exactly this size. Defined values are:

| MaxSize | Meaning |
|---------|------------|
| 0b0100 | 16 bytes. |
| 0b0101 | 32 bytes. |
| 0b0110 | 64 bytes. |
| 0b0111 | 128 bytes. |
| 0b1000 | 256 bytes. |
| 0b1001 | 512 bytes. |
| 0b1010 | 1KB. |
| 0b1011 | 2KB. |

All other values are reserved.

The values 0b0100 and 0b0101 are not permitted for an implementation.

Access to this field is **RO**.

Interval, bits [11:8]

Recommended minimum sampling interval. This provides guidance from the implementer to the smallest minimum sampling interval, N. Defined values are:

| Interval | Meaning |
|----------|---------|
| 0b0000 | 256. |
| 0b0010 | 512. |
| 0b0011 | 768. |
| 0b0100 | 1,024. |
| 0b0101 | 1,536. |
| 0b0110 | 2,048. |
| 0b0111 | 3,072. |
| 0b1000 | 4,096. |

All other values are reserved.

Access to this field is **RO**.

Bit [7]

Reserved, RES0.

FnE, bit [6]

Filtering by events, inverted. Defined values are:

| FnE | Meaning |
|-----|--|
| 0b0 | PMSNEVFR_EL1 is not implemented and PMSFCR_EL1 .FnE is RES0. |
| 0b1 | PMSNEVFR_EL1 and PMSFCR_EL1 .FnE are implemented. |

FEAT_SPEv1p2 adds the functionality identified by the value 1.

ERnd, bit [5]

Defines how the random number generator is used in determining the interval between samples, when enabled by [PMSIRR_EL1](#).RND. Defined values are:

| ERnd | Meaning |
|------|--|
| 0b0 | The random number is added at the start of the interval, and the sample is taken and a new interval started when the combined interval expires. |
| 0b1 | The random number is added and the new interval started after the interval programmed in PMSIRR_EL1 .INTERVAL expires, and the sample is taken when the random interval expires. |

Access to this field is **RO**.

LDS, bit [4]

Data source indicator for sampled load instructions. Defined values are:

| LDS | Meaning |
|-----|-------------------------------------|
| 0b0 | Loaded data source not implemented. |
| 0b1 | Loaded data source implemented. |

Access to this field is **RO**.

ArchInst, bit [3]

Architectural instruction profiling. Defined values are:

| ArchInst | Meaning |
|----------|--|
| 0b0 | Micro-op sampling implemented. |
| 0b1 | Architecture instruction sampling implemented. |

Access to this field is **RO**.

FL, bit [2]

Filtering by latency. This bit is RAO.

FT, bit [1]

Filtering by operation type. This bit is RAO.

FE, bit [0]

Filtering by events. This bit is RAO.

Accessing PMSIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMSIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSIDR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMSIDR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMSIDR_EL1;
elsif PSTATE.EL == EL3 then
    return PMSIDR_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSIRR_EL1, Sampling Interval Reload Register

The PMSIRR_EL1 characteristics are:

Purpose

Defines the interval between samples.

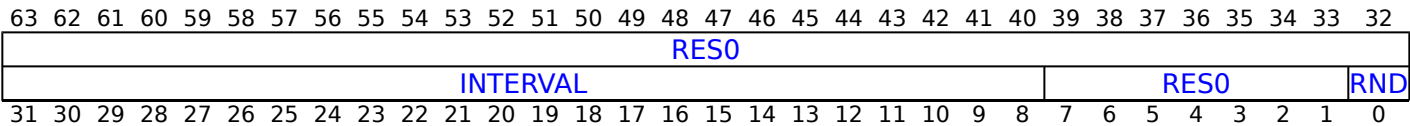
Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSIRR_EL1 are UNDEFINED.

Attributes

PMSIRR_EL1 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

INTERVAL, bits [31:8]

Bits [31:8] of the PMSICR_EL1 interval counter reload value. Software must set this to a non-zero value. If software sets this to zero, an UNKNOWN sampling interval is used. Software should set this to a value greater than the minimum indicated by PMSIDR_EL1.Interval.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:1]

Reserved, RES0.

RND, bit [0]

Controls randomization of the sampling interval.

| RND | Meaning |
|-----|--|
| 0b0 | Disable randomization of sampling interval. |
| 0b1 | Add (pseudo-)random jitter to sampling interval. |

The random number generator is not architected.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSIRR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMSIRR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSIRR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x840];
    else
        return PMSIRR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSIRR_EL1;
elsif PSTATE.EL == EL3 then
    return PMSIRR_EL1;

```

MSR PMSIRR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSIRR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            NVMem[0x840] = X[t];
        else
            PMSIRR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSIRR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMSIRR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSLATFR_EL1, Sampling Latency Filter Register

The PMSLATFR_EL1 characteristics are:

Purpose

Controls sample filtering by latency

Configuration

This register is present only when FEAT_SPE is implemented. Otherwise, direct accesses to PMSLATFR_EL1 are UNDEFINED.

Attributes

PMSLATFR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | MINLAT | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

MINLAT, bits [15:0]

Minimum latency. When [PMSFCR_EL1](#).FL is 1, defines the minimum total latency for filtered operations. Samples with a total latency less than PMSLATFR_EL1.MINLAT are not recorded.

If [PMSIDR_EL1](#).CountSize is 0b0010, PMSLATFR_EL1.MINLAT[15:12] is RES0.

This field is ignored by the PE when [PMSFCR_EL1](#).FL == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSLATFR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMSLATFR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSLATFR_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
        return NVMem[0x848];
    else
        return PMSLATFR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMSLATFR_EL1;
elsif PSTATE.EL == EL3 then
    return PMSLATFR_EL1;

```

MSR PMSLATFR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSLATFR_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            NVMem[0x848] = X[t];
        else
            PMSLATFR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSLATFR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMSLATFR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSNEVFR_EL1, Sampling Inverted Event Filter Register

The PMSNEVFR_EL1 characteristics are:

Purpose

Controls sample filtering by events. The overall filter is the logical AND of these filters. For example, if E[3] and E[5] are both set to 1, only samples that have both event 3 (Level 1 unified or data cache refill) and event 5 (TLB walk) clear are recorded.

Configuration

This register is present only when FEAT_SPEv1p2 is implemented. Otherwise, direct accesses to PMSNEVFR_EL1 are UNDEFINED.

Attributes

PMSNEVFR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|-------|--------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | |
| E[63] | E[62] | E[61] | E[60] | E[59] | E[58] | E[57] | E[56] | E[55] | E[54] | E[53] | E[52] | E[51] | E[50] | E[49] | E[48] | | | | | | |
| E[31] | E[30] | E[29] | E[28] | E[27] | E[26] | E[25] | E[24] | RAZ/WI | | | | | | E[18] | E[17] | E[16] | E[15] | E[14] | E[13] | E[12] | E[11] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | |

E[<x>], bit [x], for x = 63 to 48, 31 to 24, 15 to 12

E[<x>] is the event filter for IMPLEMENTATION DEFINED event <x>.

| E[<x>] | Meaning |
|--------|---|
| 0b0 | Event <x> is ignored. |
| 0b1 | Do not record samples that have event <x> == 1. |

An IMPLEMENTATION DEFINED event might be recorded as a multi-bit field. In this case, the corresponding bits of PMSNEVFR_EL1 define an IMPLEMENTATION DEFINED filter for the event.

This bit is ignored by the PE when [PMSFCR_EL1.FnE](#) == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When event <x> is not implemented, or filtering on event <x> is not supported, access to this field is **RAZ/WI**.

Bits [47:32]

Reserved, RAZ/WI.

Bits [23:19]

Reserved, RAZ/WI.

E[18], bit [18]**When FEAT_SVE is implemented and FEAT_SPEv1p1 is implemented:**

Not empty predicate.

| E[18] | Meaning |
|--------------|---|
| 0b0 | Empty predicate event is ignored. |
| 0b1 | Do not record samples that have the Empty predicate event == 1. |

This field is ignored by the PE when [PMSFCR_EL1](#).FnE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[17], bit [17]**When FEAT_SVE is implemented and FEAT_SPEv1p1 is implemented:**

Not partial predicate.

| E[17] | Meaning |
|--------------|---|
| 0b0 | Partial predicate event is ignored. |
| 0b1 | Do not record samples that have the Partial predicate event == 1. |

This field is ignored by the PE when [PMSFCR_EL1](#).FnE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[16], bit [16]**When FEAT_TME is implemented:**

Not transactional.

| E[16] | Meaning |
|--------------|---|
| 0b0 | Transactional event is ignored. |
| 0b1 | Do not record samples that have the Transactional event == 1. |

This field is ignored by the PE when [PMSFCR_EL1](#).FnE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

E[11], bit [11]

When FEAT_SPEv1p1 is implemented:

Aligned.

| E[11] | Meaning |
|--------------|--|
| 0b0 | Misalignment event is ignored. |
| 0b1 | Do not record samples that have the Misalignment event == 1. |

This field is ignored by the PE when [PMSFCR_EL1](#).FnE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bits [10:8]

Reserved, RAZ/WI.

E[7], bit [7]

Correctly predicted.

| E[7] | Meaning |
|-------------|--|
| 0b0 | Mispredicted event is ignored. |
| 0b1 | Do not record samples that have the Mispredicted event == 1. |

This field is ignored by the PE when [PMSFCR_EL1](#).FnE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[6], bit [6]

Taken.

| E[6] | Meaning |
|-------------|---|
| 0b0 | Not taken event is ignored. |
| 0b1 | Do not record samples that have the Not taken event == 1. |

This field is ignored by the PE when [PMSFCR_EL1](#).FnE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E[5], bit [5]

TLB hit.

| E[5] | Meaning |
|-------------|--|
| 0b0 | TLB walk event is ignored. |
| 0b1 | Do not record samples that have the TLB walk event == 1. |

This field is ignored by the PE when [PMSFCR_EL1](#).FnE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [4]

Reserved, RAZ/WI.

E[3], bit [3]

Level 1 data or unified cache hit.

| E[3] | Meaning |
|------|--|
| 0b0 | Level 1 data or unified cache refill event is ignored. |
| 0b1 | Do not record samples that have the Level 1 data or unified cache refill event == 1. |

This field is ignored by the PE when [PMSFCR_EL1](#).FnE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [2]

Reserved, RAZ/WI.

E[1], bit [1]

When the PE supports sampling of speculative instructions:

Speculative.

| E[1] | Meaning |
|------|--|
| 0b0 | Architecturally executed event is ignored. |
| 0b1 | Do not record samples that have the Architecturally executed event == 1. |

This field is ignored by the PE when [PMSFCR_EL1](#).FnE == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bit [0]

Reserved, RAZ/WI.

Accessing PMSNEVFR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMSNEVFR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.EnPMSN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.nPMSNEVFR_EL1 ==
'0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPMSN == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            return NVMem[0x850];
        else
            return PMSNEVFR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.EnPMSN == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPMSN == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMSNEVFR_EL1;
    elsif PSTATE.EL == EL3 then
        return PMSNEVFR_EL1;

```

MSR PMSNEVFR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.EnPMSN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.nPMSNEVFR_EL1 ==
'0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPMS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPMSN == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '1x1' then
            NVMem[0x850] = X[t];
        else
            PMSNEVFR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.EnPMSN == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && (MDCR_EL3.NSPB[0] == '0' || MDCR_EL3.NSPB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSPBE != SCR_EL3.NSE)) then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.EnPMSN == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMSNEVFR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        PMSNEVFR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSWINC_EL0, Performance Monitors Software Increment register

The PMSWINC_EL0 characteristics are:

Purpose

Increments a counter that is configured to count the Software increment event, event 0x00. For more information, see 'SW_INCR'.

Configuration

AArch64 System register PMSWINC_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMSWINC\[31:0\]](#).

AArch64 System register PMSWINC_EL0 bits [31:0] are architecturally mapped to External register [PMSWINC_EL0\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMSWINC_EL0 are UNDEFINED.

Attributes

PMSWINC_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:31]

Reserved, RES0.

P<n>, bit [n], for n = 30 to 0

Event counter software increment bit for [PMEVCNTR<n>_EL0](#).

If N is less than 31, then bits [30:N] are WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in [MDCR_EL2](#).HPMN. Otherwise, N is the value in [PMCR_EL0](#).N.

| P<n> | Meaning |
|------|---|
| 0b0 | No action. The write to this bit is ignored. |
| 0b1 | If PMEVCNTR<n>_EL0 is enabled and configured to count the software increment event, increments PMEVCNTR<n>_EL0 by 1. If PMEVCNTR<n>_EL0 is disabled, or not configured to count the software increment event, the write to this bit is ignored. |

Accessing PMSWINC_EL0

Accesses to this register use the following encodings in the System register encoding space:

MSR PMSWINC_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b100 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.<SW,EN> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMSWINC_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSWINC_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSWINC_EL0 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSWINC_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMSWINC_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    PMSWINC_EL0 = X[t];

```

Purpose

Configuration

Attributes

Field descriptions

Bits [63:4]

ER, bit [3]

Event counter Read. Traps EL0 access to event counters to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2.TGE](#) is 1.

In AArch64 state, trapped accesses are reported using EC syndrome value 0x18.

In AArch32 state, trapped accesses are reported using EC syndrome value 0x03.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CR, bit [2]

Cycle counter Read. Traps EL0 access to cycle counter reads to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2.TGE](#) is 1.

In AArch64 state, trapped accesses are reported using EC syndrome value 0x18.

In AArch32 state, trapped MRC accesses are reported using EC syndrome value 0x03, trapped MRRC accesses are reported using EC syndrome value 0x04.

| CR | Meaning |
|-----|--|
| 0b0 | EL0 using AArch64: EL0 read accesses to the PMCCNTR_EL0 are trapped if PMUSERENR_EL0.EN is also 0. EL0 using AArch32: EL0 read accesses to the PMCCNTR are trapped if PMUSERENR_EL0.EN is also 0. |
| 0b1 | Overrides PMUSERENR_EL0.EN and enables access to: <ul style="list-style-type: none"> • PMCCNTR_EL0 at EL0. • PMCCNTR at EL0. |

SW, bit [1]

Traps Software Increment writes to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2.TGE](#) is 1.

In AArch64 state, trapped accesses are reported using EC syndrome value 0x18.

In AArch32 state, trapped accesses are reported using EC syndrome value 0x03.

| SW | Meaning |
|-----|--|
| 0b0 | EL0 using AArch64: EL0 writes to the PMSWINC_EL0 are trapped if PMUSERENR_EL0.EN is also 0. EL0 using AArch32: EL0 writes to the PMSWINC are trapped if PMUSERENR_EL0.EN is also 0. |
| 0b1 | Overrides PMUSERENR_EL0.EN and enables access to: <ul style="list-style-type: none"> • PMSWINC_EL0 at EL0. • PMSWINC at EL0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EN, bit [0]

Traps EL0 accesses to the Performance Monitor registers to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2.TGE](#) is 1, from both Execution states as follows:

- In AArch64 state, MRS or MSR accesses to the following registers are reported using EC syndrome value 0x18:
 - [PMCR_EL0](#), [PMOVSCLR_EL0](#), [PMSELR_EL0](#), [PMCEID0_EL0](#), [PMCEID1_EL0](#), [PMCCNTR_EL0](#), [PMXEVTYPER_EL0](#), [PMXVCNTR_EL0](#), [PMCNTENSET_EL0](#), [PMCNTENCLR_EL0](#), [PMOVSSET_EL0](#), [PMEVCNTR<n>_EL0](#), [PMEVTYPER<n>_EL0](#), [PMCCFILTR_EL0](#), [PMSWINC_EL0](#).
 - If FEAT_PMUv3p4 is implemented, [PMMIR_EL1](#).
- In AArch32 state, MRC or MCR accesses to the following registers are reported using EC syndrome value 0x03:
 - [PMCR](#), [PMOVS](#), [PMSELR](#), [PMCEID0](#), [PMCEID1](#), [PMCCNTR](#), [PMXEVTYPER](#), [PMXVCNTR](#), [PMCNTENSET](#), [PMCNTENCLR](#), [PMOVSSET](#), [PMEVCNTR<n>](#), [PMEVTYPER<n>](#), [PMCCFILTR](#), [PMSWINC](#).
 - If FEAT_PMUv3p4 is implemented, [PMMIR](#).
 - If FEAT_PMUv3p1 is implemented, in AArch32 state, [PMCEID2](#), and [PMCEID3](#).
- In AArch32 state, MRRC or MCRR accesses to [PMCCNTR](#) are reported using EC syndrome value 0x04.

| EN | Meaning |
|-----|---|
| 0b0 | While at EL0, accesses to the specified registers at EL0 are trapped, unless overridden by one of PMUSERENR_EL0.{ER, CR, SW}. |
| 0b1 | While at EL0, software can access all of the specified registers. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMUSERENR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMUSERENR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1110 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMUSERENR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMUSERENR_EL0;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMUSERENR_EL0 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMUSERENR_EL0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return PMUSERENR_EL0;
elsif PSTATE.EL == EL3 then
    return PMUSERENR_EL0;

```

MSR PMUSERENR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMUSERENR_EL0 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMUSERENR_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        PMUSERENR_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    PMUSERENR_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMXEVCNTR_EL0, Performance Monitors Selected Event Count Register

The PMXEVCNTR_EL0 characteristics are:

Purpose

Reads or writes the value of the selected event counter, [PMEVCNTR<n>_EL0](#). [PMSELR_EL0](#).SEL determines which event counter is selected.

Configuration

AArch64 System register PMXEVCNTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMEVCNTR\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMXEVCNTR_EL0 are UNDEFINED.

Attributes

PMXEVCNTR_EL0 is a 64-bit register.

Field descriptions

When FEAT_PMUv3p5 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| PMEVCNTR<n> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PMEVCNTR<n> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PMEVCNTR<n>, bits [63:0]

Value of the selected event counter, [PMEVCNTR<n>_EL0](#), where n is the value stored in [PMSELR_EL0](#).SEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PMEVCNTR<n> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

PMEVCNTR<n>, bits [31:0]

Value of the selected event counter, [PMEVCNTR<n>_EL0](#), where n is the value stored in [PMSELR_EL0](#).SEL.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMXEVCNTR_EL0

If FEAT_FGT is implemented and [PMSELR_EL0.SEL](#) is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of [PMXEVCNTR_EL0](#) is as follows:

- If [PMSELR_EL0.SEL](#) selects an unimplemented event counter, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and [PMSELR_EL0.SEL](#) is greater than or equal to the number of accessible event counters, then reads and writes of [PMXEVCNTR_EL0](#) are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP
- Accesses to the register behave as if [PMSELR_EL0.SEL](#) has an UNKNOWN value less than the number of counters accessible at the current Exception level and Security state.
- If EL2 is implemented and enabled in the current Security state, and [PMSELR_EL0.SEL](#) is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Note

In EL0, an access is permitted if it is enabled by [PMUSERENR_EL0](#).{ER,EN}.

If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, [MDCR_EL2](#).HPMN identifies the number of accessible event counters. Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see [MDCR_EL2](#).HPMN.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMXEVCNTR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1101 | 0b010 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMXEVCNTR_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVCNTRn_EL0 ==
'1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMXEVCNTR_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMXEVCNTR_EL0;
    elsif PSTATE.EL == EL3 then
        return PMXEVCNTR_EL0;

```

MSR PMXEVCNTR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1101 | 0b010 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMXEVCNTR_EL0 = X[t];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVCNTRn_EL0 ==
'1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMXEVCNTR_EL0 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMXEVCNTR_EL0 = X[t];
    elsif PSTATE.EL == EL3 then
        PMXEVCNTR_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMXEVTYPERS_EL0, Performance Monitors Selected Event Type Register

The PMXEVTYPERS_EL0 characteristics are:

Purpose

When [PMSELR_EL0.SEL](#) selects an event counter, this accesses a [PMEVTYPERS<n>_EL0](#) register. When [PMSELR_EL0.SEL](#) selects the cycle counter, this accesses [PMCCFILTR_EL0](#).

Configuration

AArch64 System register PMXEVTYPERS_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMXEVTYPERS\[31:0\]](#).

This register is present only when FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMXEVTYPERS_EL0 are UNDEFINED.

Attributes

PMXEVTYPERS_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Event type register or PMCCFILTR_EL0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

When [PMSELR_EL0.SEL](#) == 31, this register accesses [PMCCFILTR_EL0](#).

Otherwise, this register accesses [PMEVTYPERS<n>_EL0](#) where n is the value in [PMSELR_EL0.SEL](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMXEVTYPERS_EL0

If FEAT_FGT is implemented, and [PMSELR_EL0.SEL](#) is not 31 and is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of [PMXEVTYPERS_EL0](#) is as follows:

- If [PMSELR_EL0.SEL](#) selects an unimplemented event counter, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented, and [PMSELR_EL0.SEL](#) is not 31 and is greater than or equal to the number of accessible event counters, then reads and writes of [PMXEVTYPERS_EL0](#) are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- Accesses to the register behave as if [PMSELR_EL0.SEL](#) has an UNKNOWN value less than the number of event counters accessible at the current Exception level and Security state.
- Accesses to the register behave as if [PMSELR_EL0.SEL](#) is 31.

- If EL2 is implemented and enabled in the current Security state, [PMSELR_EL0](#) is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Note

In EL0, an access is permitted if it is enabled by [PMUSERENR_EL0](#).EN.

If EL2 is implemented and enabled in the current Security state, in EL1 and EL0, [MDCR_EL2](#).HPMN identifies the number of accessible event counters. Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see [MDCR_EL2](#).HPMN.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, PMXEVTYPER_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1101 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMXEVTYPER_EL0;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVTYPERn_EL0 ==
'1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMXEVTYPER_EL0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return PMXEVTYPER_EL0;
    elsif PSTATE.EL == EL3 then
        return PMXEVTYPER_EL0;

```

MSR PMXEVTYPER_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1001 | 0b1101 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMXEVTYPER_EL0 = X[t];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVTYPERn_EL0 ==
'1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && MDCR_EL2.TPM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMXEVTYPER_EL0 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            PMXEVTYPER_EL0 = X[t];
    elsif PSTATE.EL == EL3 then
        PMXEVTYPER_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

REVIDR_EL1, Revision ID Register

The REVIDR_EL1 characteristics are:

Purpose

Provides implementation-specific minor revision information.

Configuration

AArch64 System register REVIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [REVIDR\[31:0\]](#).

If REVIDR_EL1 has the same value as [MIDR_EL1](#), then its contents have no significance.

Attributes

REVIDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing REVIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, REVIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0000 | 0b110 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TID1 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.REVIDR_EL1 == '1'
        then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            return REVIDR_EL1;
    elsif PSTATE.EL == EL2 then
        return REVIDR_EL1;
    elsif PSTATE.EL == EL3 then
        return REVIDR_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RGSR_EL1, Random Allocation Tag Seed Register.

The RGSR_EL1 characteristics are:

Purpose

Random Allocation Tag Seed Register.

Configuration

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to RGSR_EL1 are UNDEFINED.

When [GCR_EL1.RRND](#)==0b1, updates to RGSR_EL1 are implementation-specific.

Attributes

RGSR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|-----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | SEED | | | | | | | | | | | | | | | | RES0 | | | | TAG | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:24]

Reserved, RES0.

SEED, bits [23:8]

Seed register used for generating values returned by RandomAllocationTag().

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

TAG, bits [3:0]

Tag generated by the most recent IRG instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing RGSR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, RGSR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return RGSR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return RGSR_EL1;
elsif PSTATE.EL == EL3 then
    return RGSR_EL1;

```

MSR RGSR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RGSR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        RGSR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    RGSR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RMR_EL1, Reset Management Register (EL1)

The RMR_EL1 characteristics are:

Purpose

When this register is implemented:

- A write to the register at EL1 can request a Warm reset.
- If EL1 can use all Execution states, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

AArch64 System register RMR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [RMR\[31:0\]](#) when the highest implemented Exception level is EL1.

This register is present only when the highest implemented Exception level is EL1. Otherwise, direct accesses to RMR_EL1 are UNDEFINED.

When EL1 is the highest implemented Exception level:

- If EL1 can use all Execution states then this register must be implemented.
- If EL1 cannot use AArch32 then it is IMPLEMENTATION DEFINED whether the register is implemented.

Attributes

RMR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | RR | | AA64 | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

Bits [63:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

AA64, bit [0]

When EL1 is capable of using AArch32:

When EL1 can use AArch32, determines which Execution state the PE boots into after a Warm reset:

| AA64 | Meaning |
|------|----------|
| 0b0 | AArch32. |
| 0b1 | AArch64. |

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If EL1 can only use AArch64 state, this bit is RAO/WI.

When implemented as a RW field, this field resets to 1 on a Cold reset.

Otherwise:

Reserved, RAO/WI.

Accessing RMR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, RMR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b0000 | 0b010 |

```
if PSTATE.EL == EL1 && IsHighestEL(EL1) then
    return RMR_EL1;
else
    UNDEFINED;
```

MSR RMR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b0000 | 0b010 |

```
if PSTATE.EL == EL1 && IsHighestEL(EL1) then
    RMR_EL1 = X[t];
else
    UNDEFINED;
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RMR_EL2, Reset Management Register (EL2)

The RMR_EL2 characteristics are:

Purpose

When this register is implemented:

- A write to the register at EL2 can request a Warm reset.
- If EL2 can use all Execution states, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

AArch64 System register RMR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HRMR\[31:0\]](#) when the highest implemented Exception level is EL2.

This register is present only when the highest implemented Exception level is EL2. Otherwise, direct accesses to RMR_EL2 are UNDEFINED.

When EL2 is the highest implemented Exception level:

- If EL2 can use all Execution states then this register must be implemented.
- If EL2 cannot use AArch32 then it is IMPLEMENTATION DEFINED whether the register is implemented.

Attributes

RMR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|----|--|------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | RR | | AA64 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

Bits [63:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

AA64, bit [0]

When EL2 is capable of using AArch32:

When EL2 can use AArch32, determines which Execution state the PE boots into after a Warm reset:

| AA64 | Meaning |
|------|----------|
| 0b0 | AArch32. |
| 0b1 | AArch64. |

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If EL2 can only use AArch64 state, this bit is RAO/WI.

When implemented as a RW field, this field resets to 1 on a Cold reset.

Otherwise:

Reserved, RAO/WI.

Accessing RMR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, RMR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b0000 | 0b010 |

```
if PSTATE.EL == EL1 && EL2Enabled() && IsHighestEL(EL2) && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    return RMR_EL2;
else
    UNDEFINED;
```

MSR RMR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b0000 | 0b010 |

```
if PSTATE.EL == EL1 && EL2Enabled() && IsHighestEL(EL2) && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    RMR_EL2 = X[t];
else
    UNDEFINED;
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RMR_EL3, Reset Management Register (EL3)

The RMR_EL3 characteristics are:

Purpose

If EL3 is implemented and this register is implemented:

- A write to the register at EL3 can request a Warm reset.
- If EL3 can use all Execution states, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

AArch64 System register RMR_EL3 bits [31:0] are architecturally mapped to AArch32 System register [RMR\[31:0\]](#) when EL3 is implemented.

This register is present only when EL3 is implemented. Otherwise, direct accesses to RMR_EL3 are UNDEFINED.

When EL3 is implemented:

- If EL3 can use all Execution states then this register must be implemented.
- If EL3 cannot use AArch32, then it is IMPLEMENTATION DEFINED whether the register is implemented.

Otherwise, direct accesses to RMR_EL3 are UNDEFINED.

Attributes

RMR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | RR | | | | | | | | | | | | | | | |

Bits [63:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

AA64, bit [0]

When EL3 is capable of using AArch32:

When EL3 can use AArch32, determines which Execution state the PE boots into after a Warm reset:

| AA64 | Meaning |
|------|----------|
| 0b0 | AArch32. |
| 0b1 | AArch64. |

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If EL3 can only use AArch64 state, this bit is RAO/WI.

When implemented as a RW field, this field resets to 1 on a Cold reset.

Otherwise:

Reserved, RAO/WI.

Accessing RMR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, RMR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1100 | 0b0000 | 0b010 |

```
if PSTATE.EL == EL3 && IsHighestEL(EL3) then
    return RMR_EL3;
else
    UNDEFINED;
```

MSR RMR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1100 | 0b0000 | 0b010 |

```
if PSTATE.EL == EL3 && IsHighestEL(EL3) then
    RMR_EL3 = X[t];
else
    UNDEFINED;
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RNDR, Random Number

The RNDR characteristics are:

Purpose

Random Number. Returns a 64-bit random number which is reseeded from the True Random Number source at an IMPLEMENTATION DEFINED rate.

If the hardware returns a genuine random number, PSTATE.NZCV is set to 0b0000.

If the instruction cannot return a genuine random number in a reasonable period of time, PSTATE.NZCV is set to 0b0100 and the data value returned is 0.

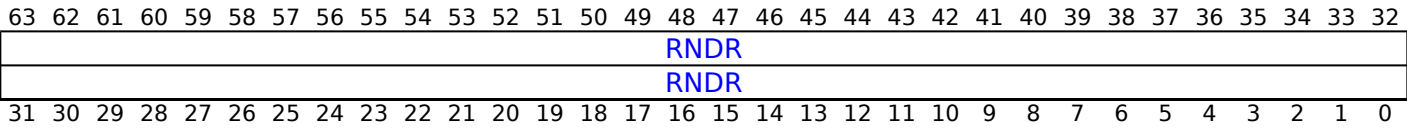
Configuration

This register is present only when FEAT_RNG is implemented or FEAT_RNG_TRAP is implemented. Otherwise, direct accesses to RNDR are UNDEFINED.

Attributes

RNDR is a 64-bit register.

Field descriptions



RNDR, bits [63:0]

Random Number. Returns a 64-bit Random Number which is reseeded from the True Random Number source at an IMPLEMENTATION DEFINED rate.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing RNDR

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, RNDR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0010 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        return RNDR;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        return RNDR;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        return RNDR;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        return RNDR;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RNDRRS, Reseeded Random Number

The RNDRRS characteristics are:

Purpose

Reseeded Random Number. Returns a 64-bit random number which is reseeded from the True Random Number source immediately before the read of the random number.

If the hardware returns a genuine random number, PSTATE.NZCV is set to 0b0000.

If the instruction cannot return a genuine random number in a reasonable period of time, PSTATE.NZCV is set to 0b0100 and the data value returned is 0.

When FEAT_RNG_TRAP is implemented and [SCR_EL3](#).TRNDR is 1, reads of this register are trapped to EL3.

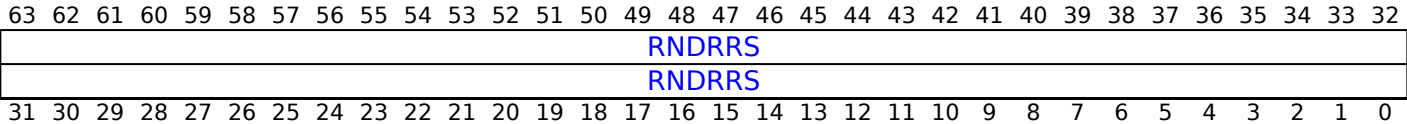
Configuration

This register is present only when FEAT_RNG is implemented or FEAT_RNG_TRAP is implemented. Otherwise, direct accesses to RNDRRS are UNDEFINED.

Attributes

RNDRRS is a 64-bit register.

Field descriptions



RNDRRS, bits [63:0]

Reseeded Random Number. Returns a 64-bit Random Number which is reseeded from the True Random Number source immediately before this read.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing RNDRRS

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, RNDRRS

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0010 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        return RNDRRS;
elsif PSTATE.EL == EL1 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        return RNDRRS;
elsif PSTATE.EL == EL2 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        return RNDRRS;
elsif PSTATE.EL == EL3 then
    if IsFeatureImplemented(FEAT_RNG_TRAP) && SCR_EL3.TRNDR == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    elsif !IsFeatureImplemented(FEAT_RNG) then
        UNDEFINED;
    else
        return RNDRRS;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

RVBAR_EL1, Reset Vector Base Address Register (if EL2 and EL3 not implemented)

The RVBAR_EL1 characteristics are:

Purpose

If EL1 is the highest Exception level implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch64 state.

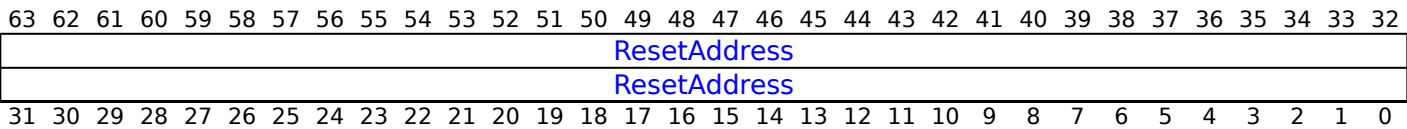
Configuration

This register is present only when the highest implemented Exception level is EL1. Otherwise, direct accesses to RVBAR_EL1 are UNDEFINED.

Attributes

RVBAR_EL1 is a 64-bit register.

Field descriptions



ResetAddress, bits [63:0]

The IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are 00, as this address must be aligned, and the address must be within the physical address size supported by the PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing RVBAR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, RVBAR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b0000 | 0b001 |

```
if PSTATE.EL == EL1 && IsHighestEL(EL1) then
    return RVBAR_EL1;
else
    UNDEFINED;
```


RVBAR_EL2, Reset Vector Base Address Register (if EL3 not implemented)

The RVBAR_EL2 characteristics are:

Purpose

If EL2 is the highest Exception level implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch64 state.

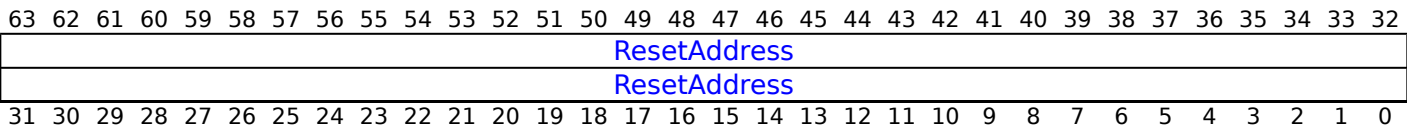
Configuration

This register is present only when the highest implemented Exception level is EL2. Otherwise, direct accesses to RVBAR_EL2 are UNDEFINED.

Attributes

RVBAR_EL2 is a 64-bit register.

Field descriptions



ResetAddress, bits [63:0]

The IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are 00, as this address must be aligned, and the address must be within the physical address size supported by the PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing RVBAR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, RVBAR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b0000 | 0b001 |

```
if PSTATE.EL == EL1 && EL2Enabled() && IsHighestEL(EL2) && HCR_EL2.NV == '1' then
    AArch64.SystemAccessTrap(EL2, 0x18);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    return RVBAR_EL2;
else
    UNDEFINED;
```


RVBAR_EL3, Reset Vector Base Address Register (if EL3 implemented)

The RVBAR_EL3 characteristics are:

Purpose

If EL3 is the highest Exception level implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch64 state.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to RVBAR_EL3 are UNDEFINED.

Only implemented if the highest Exception level implemented is EL3.

Attributes

RVBAR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ResetAddress | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ResetAddress | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ResetAddress, bits [63:0]

The IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 64-bit state. Bits[1:0] of this register are 00, as this address must be aligned, and the address must be within the physical address size supported by the PE.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing RVBAR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, RVBAR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1100 | 0b0000 | 0b001 |

```
if PSTATE.EL == EL3 && IsHighestEL(EL3) then
    return RVBAR_EL3;
else
    UNDEFINED;
```


SYS S1_<op1>_<Cn>_<Cm>_<op2>, SYSL S1_<op1>_<Cn>_<Cm>_<op2>, IMPLEMENTATION DEFINED maintenance instructions

The SYS S1_<op1>_<Cn>_<Cm>_<op2>, SYSL S1_<op1>_<Cn>_<Cm>_<op2> characteristics are:

Purpose

This area of the System instruction encoding space is reserved for IMPLEMENTATION DEFINED System instructions.

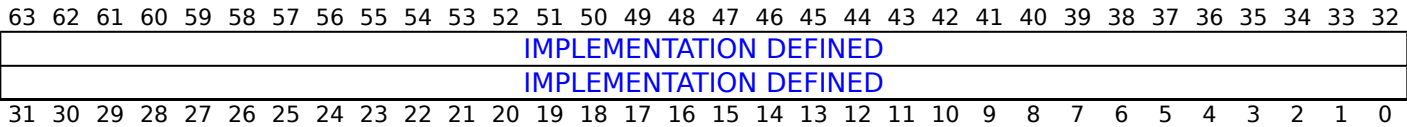
Configuration

There are no configuration notes.

Attributes

SYS S1_<op1>_<Cn>_<Cm>_<op2>, SYSL S1_<op1>_<Cn>_<Cm>_<op2> is a 64-bit System instruction.

Field descriptions



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Executing the SYS S1_<op1>_<Cn>_<Cm>_<op2>, SYSL S1_<op1>_<Cn>_<Cm>_<op2> instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

SYS #<op1>, <Cn>, <Cm>, #<op2>{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|----------|--------|---------|----------|
| 0b01 | op1[2:0] | 0b1x11 | Cm[3:0] | op2[2:0] |

```

if PSTATE.EL == EL0 then
    if SCTL_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTL_EL2.TIDCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            IMPLEMENTATION_DEFINED "SYS";
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TIDCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            IMPLEMENTATION_DEFINED "SYS";
    elsif PSTATE.EL == EL2 then
        IMPLEMENTATION_DEFINED "SYS";
    elsif PSTATE.EL == EL3 then
        IMPLEMENTATION_DEFINED "SYS";

```

SYSL <Xt>, #<op1>, <Cn>, <Cm>, #<op2>

| op0 | op1 | CRn | CRm | op2 |
|------|----------|--------|---------|----------|
| 0b01 | op1[2:0] | 0b1x11 | Cm[3:0] | op2[2:0] |

```

if PSTATE.EL == EL0 then
    if SCTL_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTL_EL2.TIDCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            IMPLEMENTATION_DEFINED "SYS";
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TIDCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            IMPLEMENTATION_DEFINED "SYS";
    elsif PSTATE.EL == EL2 then
        IMPLEMENTATION_DEFINED "SYS";
    elsif PSTATE.EL == EL3 then
        IMPLEMENTATION_DEFINED "SYS";

```

S3_<op1>_<Cn>_<Cm>_<op2>, IMPLEMENTATION DEFINED registers

The S3_<op1>_<Cn>_<Cm>_<op2> characteristics are:

Purpose

This area of the instruction set space is reserved for IMPLEMENTATION DEFINED registers.

Configuration

There are no configuration notes.

Attributes

S3_<op1>_<Cn>_<Cm>_<op2> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing S3_<op1>_<Cn>_<Cm>_<op2>

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, S3_<op1>_C<Cn>_C<Cm>_<op2>

| op0 | op1 | CRn | CRm | op2 |
|------|----------|--------|---------|----------|
| 0b11 | op1[2:0] | 0b1x11 | Cm[3:0] | op2[2:0] |

```

if PSTATE.EL == EL0 then
    if SCTL_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTL_EL2.TIDCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            IMPLEMENTATION_DEFINED "S3";
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TIDCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            IMPLEMENTATION_DEFINED "S3";
    elsif PSTATE.EL == EL2 then
        IMPLEMENTATION_DEFINED "S3";
    elsif PSTATE.EL == EL3 then
        IMPLEMENTATION_DEFINED "S3";

```

MSR S3_<op1>_C<Cn>_C<Cm>_<op2>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|----------|--------|---------|----------|
| 0b11 | op1[2:0] | 0b1x11 | Cm[3:0] | op2[2:0] |

```

if PSTATE.EL == EL0 then
    if SCTL_EL1.TIDCP == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTL_EL2.TIDCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            IMPLEMENTATION_DEFINED "S3";
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && HCR_EL2.TIDCP == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            IMPLEMENTATION_DEFINED "S3";
    elsif PSTATE.EL == EL2 then
        IMPLEMENTATION_DEFINED "S3";
    elsif PSTATE.EL == EL3 then
        IMPLEMENTATION_DEFINED "S3";

```

SCR_EL3, Secure Configuration Register

The SCR_EL3 characteristics are:

Purpose

Defines the configuration of the current Security state. It specifies:

- The Security state of EL0, EL1, and EL2. The Security state is Secure, Non-secure, or Realm.
- The Execution state at lower Exception levels.
- Whether IRQ, FIQ, SError interrupts, and External abort exceptions are taken to EL3.
- Whether various operations are trapped to EL3.

Configuration

AArch64 System register SCR_EL3 bits [31:0] can be mapped to AArch32 System register [SCR\[31:0\]](#), but this is not architecturally mandated.

This register is present only when EL3 is implemented. Otherwise, direct accesses to SCR_EL3 are UNDEFINED.

Attributes

SCR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | |
|--------|--------|-------|-------|-------|------|------|------|-------|-------|------|-----|-----|------|-------|------|-------|------|-----|-----|----|----|-------|------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 |
| RES0 | NSE | RES0 | | | | | | | | | | | | | | GPF | RES0 | | | | | EnTP2 | TRND |
| TWEDEL | TWEDEn | ECVEn | FGTEn | ATAEn | SCXT | RES0 | FIEN | NMEAE | EASEE | EEL2 | API | APK | TERR | TLORT | TWET | TWIST | RW | SIF | HCE | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 |

Bit [63]

Reserved, RES0.

NSE, bit [62]

When FEAT_RME is implemented:

This field, evaluated with SCR_EL3.NS, selects the Security state of EL2 and lower Exception levels.

For a description of the values derived by evaluating NS and NSE together, see SCR_EL3.NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [61:49]

Reserved, RES0.

GPF, bit [48]**When FEAT_RME is implemented:**

Controls the reporting of Granule protection faults at EL0, EL1 and EL2.

| GPF | Meaning |
|-----|---|
| 0b0 | This control does not cause exceptions to be routed from EL0, EL1 or EL2 to EL3. |
| 0b1 | GPFs at EL0, EL1 and EL2 are routed to EL3 and reported as Granule Protection Check exceptions. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [47:42]

Reserved, RES0.

EnTP2, bit [41]**When FEAT_SME is implemented:**

Traps instructions executed at EL2, EL1, and EL0 that access [TPIDR2_EL0](#) to EL3. The exception is reported using ESR_ELx.EC value 0x18.

| EnTP2 | Meaning |
|-------|---|
| 0b0 | This control causes execution of these instructions at EL2, EL1, and EL0 to be trapped. |
| 0b1 | This control does not cause execution of any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRNDR, bit [40]**When FEAT_RNG_TRAP is implemented:**

Controls trapping of reads of [RNDR](#) and [RNDRRS](#). The exception is reported using ESR_ELx.EC value 0x18.

| TRNDR | Meaning |
|-------|--|
| 0b0 | <p>This control does not cause RNDR and RDNRRS to be trapped.</p> <p>When FEAT_RNG is implemented:</p> <ul style="list-style-type: none"> ID_AA64ISAR0_EL1.RNDR returns the value 0b0001. <p>When FEAT_RNG is not implemented:</p> <ul style="list-style-type: none"> ID_AA64ISAR0_EL1.RNDR returns the value 0b0000. MRS reads of RNDR and RDNRRS are UNDEFINED. |
| 0b1 | <p>ID_AA64ISAR0_EL1.RNDR returns the value 0b0001.</p> <p>Any attempt to read RNDR or RNDRRS is trapped to EL3.</p> |

When FEAT_RNG is not implemented, Arm recommends that SCR_EL3.TRNDR is initialized before entering Exception levels below EL3 and not subsequently changed.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [39]

Reserved, RES0.

HXEn, bit [38]

When FEAT_HCX is implemented:

Enables access to the [HCRX_EL2](#) register at EL2 from EL3.

| HXEn | Meaning |
|------|--|
| 0b0 | Accesses at EL2 to HCRX_EL2 are trapped to EL3. Indirect reads of HCRX_EL2 return 0. |
| 0b1 | This control does not cause any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ADEn, bit [37]

When FEAT_LS64 is implemented:

Enables access to the [ACCDATA_EL1](#) register at EL1 and EL2.

| ADEn | Meaning |
|------|---|
| 0b0 | Accesses to ACCDATA_EL1 at EL1 and EL2 are trapped to EL3, unless the accesses are trapped to EL2 by the EL2 fine-grained trap. |
| 0b1 | This control does not cause accesses to ACCDATA_EL1 to be trapped. |

If the [HFGWTR_EL2](#).nACCDATA_EL1 or [HFGTR_EL2](#).nACCDATA_EL1 traps are enabled, they take priority over this trap.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [36]

When FEAT_LS64 is implemented:

Traps execution of an ST64BV0 instruction at EL0, EL1, or EL2 to EL3.

| EnAS0 | Meaning |
|-------|---|
| 0b0 | EL0 execution of an ST64BV0 instruction is trapped to EL3, unless it is trapped to EL1 by SCTLR_EL1.EnAS0 , or to EL2 by either HCRX_EL2.EnAS0 or SCTLR_EL2.EnAS0 . EL1 execution of an ST64BV0 instruction is trapped to EL3, unless it is trapped to EL2 by HCRX_EL2.EnAS0 . EL2 execution of an ST64BV0 instruction is trapped to EL3. |
| 0b1 | This control does not cause any instructions to be trapped. |

A trap of an ST64BV0 instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AMVOFFEN, bit [35]

When FEAT_AMUv1p1 is implemented:

Activity Monitors Virtual Offsets Enable.

| AMVOFFEN | Meaning |
|----------|--|
| 0b0 | Accesses to AMEVCNTVOFF0<n>_EL2 and AMEVCNTVOFF1<n>_EL2 at EL2 are trapped to EL3. Indirect reads of the virtual offset registers are zero. |
| 0b1 | Accesses to AMEVCNTVOFF0<n>_EL2 and AMEVCNTVOFF1<n>_EL2 are not affected by this field. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME, bit [34]

When FEAT_TME is implemented:

Enables access to the TSTART, TCOMMIT, TTEST and TCANCEL instructions at EL0, EL1 and EL2.

| TME | Meaning |
|-----|---|
| 0b0 | EL0, EL1 and EL2 accesses to TSTART, TCOMMIT, TTEST and TCANCEL instructions are UNDEFINED. |
| 0b1 | This control does not cause any instruction to be UNDEFINED. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEL, bits [33:30]**When FEAT_TWED is implemented:**

TWE Delay. A 4-bit unsigned number that, when SCR_EL3.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE* caused by SCR_EL3.TWE as $2^{(TWEDEL + 8)}$ cycles.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [29]**When FEAT_TWED is implemented:**

TWE Delay Enable. Enables a configurable delayed trap of the WFE* instruction caused by SCR_EL3.TWE.

Traps are reported using an ESR_ELx.EC value of 0x01.

| TWEDEn | Meaning |
|---------------|---|
| 0b0 | The delay for taking the trap is IMPLEMENTATION DEFINED. |
| 0b1 | The delay for taking the trap is at least the number of cycles defined in SCR_EL3.TWEDEL. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ECVEn, bit [28]**When FEAT_ECV is implemented:**

ECV Enable. Enables access to the [CNTPOFF_EL2](#) register.

| ECVEn | Meaning |
|--------------|---|
| 0b0 | EL2 accesses to CNTPOFF_EL2 are trapped to EL3, and the value of CNTPOFF_EL2 is treated as 0 for all purposes other than direct reads or writes to the register from EL3. |
| 0b1 | EL2 accesses to CNTPOFF_EL2 are not trapped to EL3 by this mechanism. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FGTEn, bit [27]**When FEAT_FGT is implemented:**

Fine-Grained Traps Enable. When EL2 is implemented, enables the traps to EL2 controlled by [HAFGRTR_EL2](#), [HDFGRTR_EL2](#), [HDFGWTR_EL2](#), [HFGTRTR_EL2](#), [HFGITR_EL2](#), and [HFGWTR_EL2](#), and controls access to those registers.

Note

If EL2 is not implemented but EL3 is implemented, FEAT_FGT implements the [MDCR_EL3](#).TDC traps.

| FGTEn | Meaning |
|-------|--|
| 0b0 | EL2 accesses to HAFGRTR_EL2 , HDFGRTR_EL2 , HDFGWTR_EL2 , HFGTRTR_EL2 , HFGITR_EL2 and HFGWTR_EL2 registers are trapped to EL3, and the traps to EL2 controlled by those registers are disabled. |
| 0b1 | EL2 accesses to HAFGRTR_EL2 , HDFGRTR_EL2 , HDFGWTR_EL2 , HFGTRTR_EL2 , HFGITR_EL2 and HFGWTR_EL2 registers are not trapped to EL3 by this mechanism. |

Traps caused by accesses to the fine-grained trap registers are reported using an ESR_ELx.EC value of 0x18 and its associated ISS.

Otherwise:

Reserved, RES0.

ATA, bit [26]

When FEAT_MTE2 is implemented:

Allocation Tag Access. Controls access at EL2, EL1 and EL0 to Allocation Tags.

| ATA | Meaning |
|-----|--|
| 0b0 | Access to Allocation Tags is prevented. Accesses at EL1 and EL2 to GCR_EL1 , RGSR_EL1 , TFSR_EL1 , TFSR_EL2 or TFSRE0_EL1 that are not UNDEFINED or trapped to a lower Exception level are trapped to EL3. Accesses at EL2 to TFSR_EL12 that are not UNDEFINED are trapped to EL3. |
| 0b1 | This control does not prevent access to Allocation Tags. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnSCXT, bit [25]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Enable access to the [SCXTNUM_EL2](#), [SCXTNUM_EL1](#), and [SCXTNUM_EL0](#) registers.

| EnSCXT | Meaning |
|--------|---|
| 0b0 | Accesses at EL0, EL1 and EL2 to SCXTNUM_EL0 , SCXTNUM_EL1 , or SCXTNUM_EL2 registers are trapped to EL3 if they are not trapped by a higher priority exception, and the values of these registers are treated as 0. |
| 0b1 | This control does not cause any accesses to be trapped, or register values to be treated as 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [24:22]

Reserved, RES0.

FIEN, bit [21]

When FEAT_RASv1p1 is implemented:

Fault Injection enable. Trap accesses to the registers [ERXPFPCDN_EL1](#), [ERXPFPGCTL_EL1](#), and [ERXPFPGF_EL1](#) from EL1 and EL2 to EL3, reported using an ESR_ELx.EC value of 0x18.

| FIEN | Meaning |
|------|--|
| 0b0 | Accesses to the specified registers from EL1 and EL2 generate a Trap exception to EL3. |
| 0b1 | This control does not cause any instructions to be trapped. |

If EL3 is not implemented, the Effective value of SCR_EL3.FIEN is 0b1.

If [ERRIDR_EL1](#).NUM is zero, meaning no error records are implemented, or no error record accessible using System registers is owned by a node that implements the RAS Common Fault Injection Model Extension, then this bit might be RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMEA, bit [20]

When FEAT_DoubleFault is implemented:

Non-maskable External Aborts. When [SCR_EL3](#).EA == 1, controls whether PSTATE.A masks SError interrupts at EL3.

| NMEA | Meaning |
|------|---|
| 0b0 | If SCR_EL3 .EA == 1, asserted SError interrupts are not taken at EL3 if PSTATE.A == 1. |
| 0b1 | If SCR_EL3 .EA == 1, asserted SError interrupts are taken at EL3 regardless of the value of PSTATE.A. |

When SCR_EL3.EA == 0:

- Asserted SError interrupts are not taken at EL3 regardless of the value of PSTATE.A and this field.
- This field is ignored and its Effective value is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

EASE, bit [19]**When FEAT_DoubleFault is implemented:**

External aborts to SError interrupt vector.

| EASE | Meaning |
|-------------|---|
| 0b0 | Synchronous External abort exceptions taken to EL3 are taken to the appropriate synchronous exception vector offset from VBAR_EL3 . |
| 0b1 | Synchronous External abort exceptions taken to EL3 are taken to the appropriate SError interrupt vector offset from VBAR_EL3 . |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

EEL2, bit [18]**When FEAT_SEL2 is implemented:**

Secure EL2 Enable.

| EEL2 | Meaning |
|-------------|---|
| 0b0 | All behaviors associated with Secure EL2 are disabled. All registers, including timer registers, defined by FEAT_SEL2 are UNDEFINED, and those timers are disabled. |
| 0b1 | All behaviors associated with Secure EL2 are enabled. |

When the value of this bit is 1, then:

- When SCR_EL3.NS == 0, the SCR_EL3.RW bit is treated as 1 for all purposes other than reading or writing the register.
- If Secure EL1 is using AArch32, then any of the following operations, executed in Secure EL1, is trapped to Secure EL2, using the EC value of [ESR_EL2](#).EC == 0x3 :
 - A read or write of the [SCR](#).
 - A read or write of the [NSACR](#).
 - A read or write of the [MVBAR](#).
 - A read or write of the [SDCR](#).
 - Execution of an ATS12NSO** instruction.
- If Secure EL1 is using AArch32, then any of the following operations, executed in Secure EL1, is trapped to Secure EL2 using the EC value of [ESR_EL2](#).EC == 0x0 :
 - Execution of an SRS instruction that uses R13_mon.
 - Execution of an MRS (Banked register) or MSR (Banked register) instruction that would access [SPSR_mon](#), R13_mon, or R14_mon.

Note

If the Effective value of SCR_EL3.EEL2 is 0, then these operations executed in Secure EL1 using AArch32 are trapped to EL3.

A Secure only implementation that does not implement EL3 but implements EL2, behaves as if SCR_EL3.EEL2 == 1.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

API, bit [17]**When FEAT_SEL2 is implemented and FEAT_PAuth is implemented:**

Controls the use of the following instructions related to Pointer Authentication. Traps are reported using an ESR_ELx.EC value of 0x09:

- PACGA, which is always enabled.
- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZB, RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ, ERETAA, ERETAB, LDRAA and LDRAB when:
 - In EL0, when [HCR_EL2.TGE](#) == 0 or [HCR_EL2.E2H](#) == 0, and the associated [SCTLR_EL1.En<N><M>](#) == 1.
 - In EL0, when [HCR_EL2.TGE](#) == 1 and [HCR_EL2.E2H](#) == 1, and the associated [SCTLR_EL2.En<N><M>](#) == 1.
 - In EL1, when the associated [SCTLR_EL1.En<N><M>](#) == 1.
 - In EL2, when the associated [SCTLR_EL2.En<N><M>](#) == 1.

| API | Meaning |
|-----|--|
| 0b0 | The use of any instruction related to pointer authentication in any Exception level except EL3 when the instructions are enabled are trapped to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.API bit. |
| 0b1 | This control does not cause any instructions to be trapped. |

An instruction is trapped only if Pointer Authentication is enabled for that instruction, for more information, see 'System register control of pointer authentication'.

Note

If FEAT_PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_SEL2 is not implemented and FEAT_PAuth is implemented:

Controls the use of instructions related to Pointer Authentication:

- PACGA.
- AUTDA, AUTDB, AUTDZA, AUTDZB, AUTIA, AUTIA1716, AUTIASP, AUTIAZ, AUTIB, AUTIB1716, AUTIBSP, AUTIBZ, AUTIZA, AUTIZB, PACDA, PACDB, PACDZA, PACDZB, PACIA, PACIA1716, PACIASP, PACIAZ, PACIB, PACIB1716, PACIBSP, PACIBZ, PACIZA, PACIZB, RETAA, RETAB, BRAA, BRAB, BLRAA, BLRAB, BRAAZ, BRABZ, BLRAAZ, BLRABZ, ERETAA, ERETAB, LDRAA and LDRAB when:
 - In Non-secure EL0, when [HCR_EL2.TGE](#) == 0 or [HCR_EL2.E2H](#) == 0, and the associated [SCTLR_EL1.En<N><M>](#) == 1.
 - In Non-secure EL0, when [HCR_EL2.TGE](#) == 1 and [HCR_EL2.E2H](#) == 1, and the associated [SCTLR_EL2.En<N><M>](#) == 1.
 - In Secure EL0, when the associated [SCTLR_EL1.En<N><M>](#) == 1.
 - In Secure or Non-secure EL1, when the associated [SCTLR_EL1.En<N><M>](#) == 1.
 - In EL2, when the associated [SCTLR_EL2.En<N><M>](#) == 1.

| API | Meaning |
|-----|--|
| 0b0 | The use of any instruction related to pointer authentication in any Exception level except EL3 when the instructions are enabled are trapped to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.API bit. |
| 0b1 | This control does not cause any instructions to be trapped. |

Note

If FEAT_PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

APK, bit [16]

When FEAT_PAuth is implemented:

Trap registers holding "key" values for Pointer Authentication. Traps accesses to the following registers, using an ESR_ELx.EC value of 0x18, from EL1 or EL2 to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.APK bit or other traps:

- [APIAKeyLo_EL1](#), [APIAKeyHi_EL1](#), [APIBKeyLo_EL1](#), [APIBKeyHi_EL1](#).
- [APDAKeyLo_EL1](#), [APDAKeyHi_EL1](#), [APDBKeyLo_EL1](#), [APDBKeyHi_EL1](#).
- [APGAKeyLo_EL1](#), and [APGAKeyHi_EL1](#).

| APK | Meaning |
|-----|--|
| 0b0 | Access to the registers holding "key" values for pointer authentication from EL1 or EL2 are trapped to EL3 unless they are trapped to EL2 as a result of the HCR_EL2.APK bit or other traps. |
| 0b1 | This control does not cause any instructions to be trapped. |

For more information, see 'System register control of pointer authentication'.

Note

If FEAT_PAuth is implemented but EL3 is not implemented, the system behaves as if this bit is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TERR, bit [15]

When FEAT_RAS is implemented:

Trap Error record accesses. Accesses to the RAS ERR* and RAS ERX* registers from EL1 and EL2 to EL3 are trapped as follows:

- Accesses from EL1 and EL2 using AArch64 to the following registers are trapped and reported using an ESR_ELx.EC value of 0x18:
 - [ERRIDR_EL1](#), [ERRSELR_EL1](#), [ERXADDR_EL1](#), [ERXCTLR_EL1](#), [ERXFR_EL1](#), [ERXMISC0_EL1](#), [ERXMISC1_EL1](#), and [ERXSTATUS_EL1](#).
- If FEAT_RASv1p1 is implemented, accesses from EL1 and EL2 using AArch64 to [ERXMISC2_EL1](#), and [ERXMISC3_EL1](#), are trapped and reported using an ESR_ELx.EC value of 0x18.

- Accesses from EL1 and EL2 using AArch32, to the following registers are trapped and reported using an ESR_ELx.EC value of 0x03:
 - [ERRIDR](#), [ERRSELR](#), [ERXADDR](#), [ERXADDR2](#), [ERXCTLR](#), [ERXCTLR2](#), [ERXFR](#), [ERXFR2](#), [ERXMISC0](#), [ERXMISC1](#), [ERXMISC2](#), [ERXMISC3](#), and [ERXSTATUS](#).
- If FEAT_RASv1p1 is implemented, accesses from EL1 and EL2 using AArch32 to the following registers are trapped and reported using an ESR_ELx.EC value of 0x03:
 - [ERXMISC4](#), [ERXMISC5](#), [ERXMISC6](#), and [ERXMISC7](#).

| TERR | Meaning |
|-------------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Accesses to the specified registers from EL1 and EL2 generate a Trap exception to EL3. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TLOR, bit [14]

When FEAT_LOR is implemented:

Trap LOR registers. Traps accesses to the [LORSA_EL1](#), [LOREA_EL1](#), [LORN_EL1](#), [LORC_EL1](#), and [LORID_EL1](#) registers from EL1 and EL2 to EL3, unless the access has been trapped to EL2.

| TLOR | Meaning |
|-------------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL1 and EL2 accesses to the LOR registers that are not UNDEFINED are trapped to EL3, unless it is trapped HCR_EL2.TLOR . |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWE, bit [13]

Traps EL2, EL1, and EL0 execution of WFE instructions to EL3, from any Security state and both Execution states, reported using an ESR_ELx.EC value of 0x01.

When FEAT_WFxF is implemented, this trap also applies to the WFET instruction.

| TWE | Meaning |
|------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any attempt to execute a WFE instruction at any Exception level lower than EL3 is trapped to EL3, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWE , HCR.TWE , SCTLR_EL1.nTWE , SCTLR_EL2.nTWE , or HCR_EL2.TWE . |

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

For more information about when WFE instructions can cause the PE to enter a low-power state, see 'Wait for Event mechanism and Send event'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TWI, bit [12]

Traps EL2, EL1, and EL0 execution of WFI instructions to EL3, from any Security state and both Execution states, reported using an ESR_ELx.EC value of 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFIT instruction.

| TWI | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any attempt to execute a WFI instruction at any Exception level lower than EL3 is trapped to EL3, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWI , HCR.TWI , SCTLR_EL1.nTWI , SCTLR_EL2.nTWI , or HCR_EL2.TWI . |

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

For more information about when WFI instructions can cause the PE to enter a low-power state, see 'Wait for Interrupt'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ST, bit [11]

Traps Secure EL1 accesses to the Counter-timer Physical Secure timer registers to EL3, from AArch64 state only, reported using an ESR_ELx.EC value of 0x18.

| ST | Meaning |
|-----|---|
| 0b0 | Secure EL1 using AArch64 accesses to the CNTPS_TVAL_EL1 , CNTPS_CTL_EL1 , and CNTPS_CVAL_EL1 are trapped to EL3 when Secure EL2 is disabled. If Secure EL2 is enabled, the behavior is as if the value of this field was 0b1. |
| 0b1 | This control does not cause any instructions to be trapped. |

Note

Accesses to the Counter-timer Physical Secure timer registers are always enabled at EL3. These registers are not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RW, bit [10]

When EL1 is capable of using AArch32 or EL2 is capable of using AArch32:

Execution state control for lower Exception levels.

| RW | Meaning |
|-----|--|
| 0b0 | Lower levels are all AArch32. |
| 0b1 | The next lower level is AArch64. If EL2 is present: <ul style="list-style-type: none"> EL2 is AArch64. EL2 controls EL1 and EL0 behaviors. If EL2 is not present: <ul style="list-style-type: none"> EL1 is AArch64. EL0 is determined by the Execution state described in the current process state when executing at EL0. |

If AArch32 state is supported by the implementation at EL1, SCR_EL3.NS == 1 and AArch32 state is not supported by the implementation at EL2, the Effective value of this bit is 1.

If AArch32 state is supported by the implementation at EL1, FEAT_SEL2 is implemented and SCR_EL3.{EEL2, NS} == {1, 0}, the Effective value of this bit is 1.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAO/WI.

SIF, bit [9]

When FEAT_SEL2 is implemented:

Secure instruction fetch. When the PE is in Secure state, this bit disables instruction fetch from memory marked in the first stage of translation as being Non-secure.

| SIF | Meaning |
|-----|--|
| 0b0 | Secure state instruction fetches from memory marked in the first stage of translation as being Non-secure are permitted. |
| 0b1 | Secure state instruction fetches from memory marked in the first stage of translation as being Non-secure are not permitted. |

When FEAT_PAN3 is implemented, it is IMPLEMENTATION DEFINED whether SCR_EL3.SIF is also used to determine instruction access permission for the purpose of PAN.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Secure instruction fetch. When the PE is in Secure state, this bit disables instruction fetch from Non-secure memory.

| SIF | Meaning |
|-----|--|
| 0b0 | Secure state instruction fetches from Non-secure memory are permitted. |
| 0b1 | Secure state instruction fetches from Non-secure memory are not permitted. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HCE, bit [8]

Hypervisor Call instruction enable. Enables HVC instructions at EL3 and, if EL2 is enabled in the current Security state, at EL2 and EL1, in both Execution states, reported using an ESR_ELx.EC value of 0x00.

| HCE | Meaning |
|-----|--|
| 0b0 | HVC instructions are UNDEFINED. |
| 0b1 | HVC instructions are enabled at EL3, EL2, and EL1. |

Note

HVC instructions are always UNDEFINED at EL0 and, if Secure EL2 is disabled, at Secure EL1. Any resulting exception is taken from the current Exception level to the current Exception level.

If EL2 is not implemented, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SMD, bit [7]

Secure Monitor Call disable. Disables SMC instructions at EL1 and above, from any Security state and both Execution states, reported using an ESR_ELx.EC value of 0x00.

| SMD | Meaning |
|-----|---|
| 0b0 | SMC instructions are enabled at EL3, EL2 and EL1. |
| 0b1 | SMC instructions are UNDEFINED. |

Note

SMC instructions are always UNDEFINED at EL0. Any resulting exception is taken from the current Exception level to the current Exception level.

If [HCR_EL2.TSC](#) or [HCR.TSC](#) traps attempted EL1 execution of SMC instructions to EL2, that trap has priority over this disable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

Bits [5:4]

Reserved, RES1.

EA, bit [3]

External Abort and SError interrupt routing.

| EA | Meaning |
|-----------|--|
| 0b0 | When executing at Exception levels below EL3, External aborts and SError interrupts are not taken to EL3. In addition, when executing at EL3: <ul style="list-style-type: none"> • SError interrupts are not taken. • External aborts are taken to EL3. |
| 0b1 | When executing at any Exception level, External aborts and SError interrupts are taken to EL3. |

For more information, see 'Asynchronous exception routing'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FIQ, bit [2]

Physical FIQ Routing.

| FIQ | Meaning |
|------------|--|
| 0b0 | When executing at Exception levels below EL3, physical FIQ interrupts are not taken to EL3. When executing at EL3, physical FIQ interrupts are not taken. |
| 0b1 | When executing at any Exception level, physical FIQ interrupts are taken to EL3. |

For more information, see 'Asynchronous exception routing'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRQ, bit [1]

Physical IRQ Routing.

| IRQ | Meaning |
|------------|--|
| 0b0 | When executing at Exception levels below EL3, physical IRQ interrupts are not taken to EL3. When executing at EL3, physical IRQ interrupts are not taken. |
| 0b1 | When executing at any Exception level, physical IRQ interrupts are taken to EL3. |

For more information, see 'Asynchronous exception routing'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [0]

When FEAT_RME is implemented:

Non-secure bit. This field is used in combination with SCR_EL3.NSE to select the Security state of EL2 and lower Exception levels.

| NSE | NS | Meaning |
|------------|-----------|----------------|
| 0b0 | 0b0 | Secure. |
| 0b0 | 0b1 | Non-secure. |
| 0b1 | 0b0 | Reserved. |
| 0b1 | 0b1 | Realm. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Non-secure bit.

| NS | Meaning |
|-----|---|
| 0b0 | Indicates that EL0 and EL1 are in Secure state. |
| 0b1 | Indicates that Exception levels lower than EL3 are in Non-secure state, so memory accesses from those Exception levels cannot access Secure memory. |

When $\text{SCR_EL3}\{EEL2, NS\} == \{1, 0\}$, then EL2 is using AArch64 and in Secure state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SCR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0001 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SCR_EL3;
```

MSR SCR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0001 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCR_EL3 = X[t];
```

SCTLR_EL1, System Control Register (EL1)

The SCTLR_EL1 characteristics are:

Purpose

Provides top level control of the system, including its memory system, at EL1 and EL0.

Configuration

AArch64 System register SCTLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [SCTLR\[31:0\]](#).

Attributes

SCTLR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | |
|-------|-----------|--------|--------|------|------|-------|-------|-------|-----|-------|-------|------|------|--------|------|-----|-----|--------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 |
| TIDCP | SPINTMASK | NMI | EnTP2 | RES0 | EPAN | EnALS | EnAS0 | EnASR | TME | TME0 | TMT | TMT0 | | TWEDEL | | | | TWEDEL |
| EnIA | EnIB | LSMAOE | nTLSMD | EnDA | UCI | EE | EOE | SPAN | EIS | IESBT | TSCXT | WXN | nTWE | RES0 | nTWI | UCT | DZE | EnD |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 |

TIDCP, bit [63]

When FEAT_TIDCP1 is implemented:

Trap IMPLEMENTATION DEFINED functionality. When [HCR_EL2](#).{E2H, TGE} != {1, 1}, traps EL0 accesses to the encodings reserved for IMPLEMENTATION DEFINED functionality to EL1.

| TIDCP | Meaning |
|-------|--|
| 0b0 | No instructions accessing the System register or System instruction spaces are trapped by this mechanism. |
| 0b1 | Instructions accessing the following System register or System instruction spaces are trapped to EL1 by this mechanism: <ul style="list-style-type: none"> In AArch64 state, EL0 access to the encodings in the following reserved encoding spaces are trapped and reported using EC syndrome 0x18: <ul style="list-style-type: none"> IMPLEMENTATION DEFINED System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}. IMPLEMENTATION DEFINED System registers, which are accessed using MRS and MSR with the S3_<op1>_<Cn>_<Cm>_<op2> register name. In AArch32 state, EL0 MCR and MRC access to the following encodings are trapped and reported using EC syndrome 0x03: <ul style="list-style-type: none"> All coproc==p15, CRn==c9, opc1 == {0-7}, CRm == {c0-c2, c5-c8}, opc2 == {0-7}. All coproc==p15, CRn==c10, opc1 == {0-7}, CRm == {c0, c1, c4, c8}, opc2 == {0-7}. All coproc==p15, CRn==c11, opc1 == {0-7}, CRm == {c0-c8, c15}, opc2 == {0-7}. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPINTMASK, bit [62]**When FEAT_NMI is implemented:**

Superpriority Interrupt Mask enable. When SCTLR_EL1.NMI is 1, controls the value of PSTATE.ALLINT on taking an exception to EL1.

| SPINTMASK | Meaning |
|-----------|--|
| 0b0 | PSTATE.ALLINT is set to 1 on taking an exception to EL1. |
| 0b1 | PSTATE.ALLINT is set to 0 on taking an exception to EL1. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMI, bit [61]**When FEAT_NMI is implemented:**

Non-maskable Interrupt enable. Enables support for IRQ and FIQ interrupts with Superpriority, and determines additional masking behavior of the PSTATE.I and PSTATE.F flags.

| NMI | Meaning |
|-----|--|
| 0b0 | The behaviour of PSTATE.I and PSTATE.F is unchanged. IRQ and FIQ interrupts with Superpriority have no effect on interrupts that are targeted at EL1. |
| 0b1 | IRQ and FIQ interrupts can be marked as having Superpriority as an additional attribute, and additional Superpriority masking behavior is determined by PSTATE.ALLINT. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to 0.

Otherwise:

Reserved, RES0.

EnTP2, bit [60]**When FEAT_SME is implemented:**

Traps instructions executed at EL0 that access [TPIDR2_EL0](#) to EL1, or to EL2 when EL2 is implemented and enabled for the current Security state and [HCR_EL2.TGE](#) is 1. The exception is reported using ESR_ELx.EC value 0x18.

| EnTP2 | Meaning |
|-------|---|
| 0b0 | This control causes execution of these instructions at EL0 to be trapped. |
| 0b1 | This control does not cause execution of any instructions to be trapped. |

If FEAT_VHE is implemented, EL2 is implemented and enabled in the current Security state, and [HCR_EL2.{E2H, TGE}](#) == {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [59:58]

Reserved, RES0.

EPAN, bit [57]

When FEAT_PAN3 is implemented:

Enhanced Privileged Access Never. When PSTATE.PAN is 1, determines whether an EL1 data access to a page with stage 1 EL0 instruction access permission generates a Permission fault as a result of the Privileged Access Never mechanism.

| EPAN | Meaning |
|------|---|
| 0b0 | No additional Permission faults are generated by this mechanism. |
| 0b1 | An EL1 data access to a page with stage 1 EL0 data access permission or stage 1 EL0 instruction access permission generates a Permission fault. Any speculative data accesses that would generate a Permission fault if the accesses were not speculative will not cause an allocation into a cache. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnALS, bit [56]

When FEAT_LS64 is implemented:

When [HCR_EL2](#).{E2H, TGE} != {1, 1}, traps execution of an LD64B or ST64B instruction at EL0 to EL1.

| EnALS | Meaning |
|-------|--|
| 0b0 | Execution of an LD64B or ST64B instruction at EL0 is trapped to EL1. |
| 0b1 | This control does not cause any instructions to be trapped. |

A trap of an LD64B or ST64B instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000002.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [55]

When FEAT_LS64 is implemented:

When [HCR_EL2](#).{E2H, TGE} != {1, 1}, traps execution of an ST64BV0 instruction at EL0 to EL1.

| EnAS0 | Meaning |
|-------|---|
| 0b0 | Execution of an ST64BV0 instruction at EL0 is trapped to EL1. |
| 0b1 | This control does not cause any instructions to be trapped. |

A trap of an ST64BV0 instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnASR, bit [54]

When FEAT_LS64 is implemented:

When [HCR_EL2](#).{E2H, TGE} != {1, 1}, traps execution of an ST64BV instruction at EL0 to EL1.

| EnASR | Meaning |
|-------|--|
| 0b0 | Execution of an ST64BV instruction at EL0 is trapped to EL1. |
| 0b1 | This control does not cause any instructions to be trapped. |

A trap of an ST64BV instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000000.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME, bit [53]

When FEAT_TME is implemented:

Enables the Transactional Memory Extension at EL1.

| TME | Meaning |
|-----|--|
| 0b0 | Any attempt to execute a TSTART instruction at EL1 is trapped to EL1, unless HCR_EL2 .TME or SCR_EL3 .TME causes TSTART instructions to be UNDEFINED at EL1. |
| 0b1 | This control does not cause any TSTART instruction to be trapped. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME0, bit [52]**When FEAT_TME is implemented:**

Enables the Transactional Memory Extension at EL0.

| TME0 | Meaning |
|-------------|--|
| 0b0 | Any attempt to execute a TSTART instruction at EL0 is trapped to EL1, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL0. |
| 0b1 | This control does not cause any TSTART instruction to be trapped. |

If FEAT_VHE is implemented, EL2 is implemented and enabled in the current Security state, and [HCR_EL2](#).{E2H, TGE} == {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT, bit [51]**When FEAT_TME is implemented:**

Forces a trivial implementation of the Transactional Memory Extension at EL1.

| TMT | Meaning |
|------------|---|
| 0b0 | This control does not cause any TSTART instruction to fail. |
| 0b1 | When the TSTART instruction is executed at EL1, the transaction fails with a TRIVIAL failure cause. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT0, bit [50]**When FEAT_TME is implemented:**

Forces a trivial implementation of the Transactional Memory Extension at EL0.

| TMT0 | Meaning |
|-------------|---|
| 0b0 | This control does not cause any TSTART instruction to fail. |
| 0b1 | When the TSTART instruction is executed at EL0, the transaction fails with a TRIVIAL failure cause. |

If FEAT_VHE is implemented, EL2 is implemented and enabled in the current Security state, and [HCR_EL2](#).{E2H, TGE} == {1, 1}, this field has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEL, bits [49:46]**When FEAT_TWED is implemented:**

TWE Delay. A 4-bit unsigned number that, when SCTLR_EL1.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE* caused by SCTLR_EL1.nTWE as $2^{(TWEDEL + 8)}$ cycles.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [45]**When FEAT_TWED is implemented:**

TWE Delay Enable. Enables a configurable delayed trap of the WFE* instruction caused by SCTLR_EL1.nTWE.

| TWEDEn | Meaning |
|---------------|---|
| 0b0 | The delay for taking the trap is IMPLEMENTATION DEFINED. |
| 0b1 | The delay for taking the trap is at least the number of cycles defined in SCTLR_EL1.TWEDEL. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DSSBS, bit [44]**When FEAT_SSBS is implemented:**

Default PSTATE.SSBS value on Exception Entry.

| DSSBS | Meaning |
|--------------|---|
| 0b0 | PSTATE.SSBS is set to 0 on an exception to EL1. |
| 0b1 | PSTATE.SSBS is set to 1 on an exception to EL1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

ATA, bit [43]**When FEAT_MTE2 is implemented:**

Allocation Tag Access in EL1. When [SCR_EL3.ATA](#)=1 and [HCR_EL2.ATA](#)=1, controls EL1 access to Allocation Tags.

| ATA | Meaning |
|-----|--|
| 0b0 | Access to Allocation Tags is prevented. |
| 0b1 | This control does not prevent access to Allocation Tags. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA0, bit [42]**When FEAT_MTE2 is implemented:**

Allocation Tag Access in EL0. When [SCR_EL3.ATA](#)=1, [HCR_EL2.ATA](#)=1, and [HCR_EL2](#).{E2H, TGE} != {1, 1}, controls EL0 access to Allocation Tags.

| ATA0 | Meaning |
|------|--|
| 0b0 | Access to Allocation Tags is prevented. |
| 0b1 | This control does not prevent access to Allocation Tags. |

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF, bits [41:40]**When FEAT_MTE2 is implemented:**

Tag Check Fault in EL1. Controls the effect of Tag Check Faults due to Loads and Stores in EL1.

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

| TCF | Meaning | Applies when |
|------|--|-------------------------------|
| 0b00 | Tag Check Faults have no effect on the PE. | When FEAT_MTE3 is implemented |
| 0b01 | Tag Check Faults cause a synchronous exception. | |
| 0b10 | Tag Check Faults are asynchronously accumulated. | |
| 0b11 | Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes. | |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF0, bits [39:38]**When FEAT_MTE2 is implemented:**

Tag Check Fault in EL0. When [HCR_EL2](#).{E2H,TGE} != {1,1}, controls the effect of Tag Check Faults due to Loads and Stores in EL0.

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

Note

Software may change this control bit on a context switch.

| TCF0 | Meaning | Applies when |
|------|--|-------------------------------|
| 0b00 | Tag Check Faults have no effect on the PE. | |
| 0b01 | Tag Check Faults cause a synchronous exception. | |
| 0b10 | Tag Check Faults are asynchronously accumulated. | |
| 0b11 | Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes. | When FEAT_MTE3 is implemented |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITFSB, bit [37]**When FEAT_MTE2 is implemented:**

When synchronous exceptions are not being generated by Tag Check Faults, this field controls whether on exception entry into EL1, all Tag Check Faults due to instructions executed before exception entry, that are reported asynchronously, are synchronized into [TFSRE0_EL1](#) and [TFSR_EL1](#) registers.

| ITFSB | Meaning |
|-------|--|
| 0b0 | Tag Check Faults are not synchronized on entry to EL1. |
| 0b1 | Tag Check Faults are synchronized on entry to EL1. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT1, bit [36]**When FEAT_BT1 is implemented:**

PAC Branch Type compatibility at EL1.

| BT1 | Meaning |
|-----|--|
| 0b0 | When the PE is executing at EL1, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11. |
| 0b1 | When the PE is executing at EL1, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT0, bit [35]**When FEAT_BT1 is implemented:**

PAC Branch Type compatibility at EL0.

| BT0 | Meaning |
|-----|--|
| 0b0 | When the PE is executing at EL0, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11. |
| 0b1 | When the PE is executing at EL0, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11. |

When the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, the value of SCTLR_EL1.BT0 has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [34]

Reserved, RES0.

MSCEn, bit [33]**When FEAT_MOPS is implemented and (HCR_EL2.E2H == 0 or HCR_EL2.TGE == 0):**

MemCpy and MemSet instructions Enable. Enables execution of the MemCpy and MemSet instructions at EL0.

| MSCEn | Meaning |
|-------|--|
| 0b0 | Execution of the MemCpy and MemSet instructions is UNDEFINED at EL0. |
| 0b1 | This control does not cause any instructions to be UNDEFINED. |

When FEAT_MOPS is implemented and [HCR_EL2](#).{E2H, TGE} is {1, 1}, the Effective value of this bit is 0b1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CMOW, bit [32]

When FEAT_CMOW is implemented:

Controls cache maintenance instruction permission for the following instructions executed at EL0.

- [IC IVAU](#), [DC CIVAC](#), [DC CIGDVAC](#) and [DC CIGVAC](#).

| CMOW | Meaning |
|------|--|
| 0b0 | These instructions executed at EL0 with stage 1 read permission, but without stage 1 write permission, do not generate a stage 1 permission fault. |
| 0b1 | If enabled as a result of SCTLR_EL1 .UCI==1, these instructions executed at EL0 with stage 1 read permission, but without stage 1 write permission, generate a stage 1 permission fault. |

When AArch64.HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

For this control, stage 1 has write permission if all of the following apply:

- AP[2] is 0 or DBM is 1 in the stage 1 descriptor.
- Where APTable is in use, APTable[1] is 0 for all levels of the translation table.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIA, bit [31]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication (using the APIAKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

For more information, see 'System register control of pointer authentication'.

| EnIA | Meaning |
|------|---|
| 0b0 | Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is not enabled. |
| 0b1 | Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is enabled. |

Note

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. Specifically, when the field is 1, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]**When FEAT_PAuth is implemented:**

Controls enabling of pointer authentication (using the APIBKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

For more information, see 'System register control of pointer authentication'.

| EnIB | Meaning |
|------|---|
| 0b0 | Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is not enabled. |
| 0b1 | Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is enabled. |

Note

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. Specifically, when the field is 1, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LSMAOE, bit [29]**When FEAT_LSMAOC is implemented:**

Load Multiple and Store Multiple Atomicity and Ordering Enable.

| LSMAOE | Meaning |
|--------|---|
| 0b0 | For all memory accesses at EL0, A32 and T32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered. |
| 0b1 | The ordering and interrupt behavior of A32 and T32 Load Multiple and Store Multiple at EL0 is as defined for Armv8.0. |

This bit is permitted to be cached in a TLB.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

nTLSMD, bit [28]**When FEAT_LSMAOC is implemented:**

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

| nTLSMD | Meaning |
|---------------|--|
| 0b0 | All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault. |
| 0b1 | All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped. |

This bit is permitted to be cached in a TLB.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnDA, bit [27]**When FEAT_PAuth is implemented:**

Controls enabling of pointer authentication (using the APDAKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

For more information, see 'System register control of pointer authentication'.

| EnDA | Meaning |
|-------------|--|
| 0b0 | Pointer authentication (using the APDAKey_EL1 key) of data addresses is not enabled. |
| 0b1 | Pointer authentication (using the APDAKey_EL1 key) of data addresses is enabled. |

Note

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. Specifically, when the field is 1, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UCI, bit [26]

Traps EL0 execution of cache maintenance instructions, to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2.TGE](#) is 1, from AArch64 state only, reported using an [ESR_ELx.EC](#) value of 0x18.

This applies to [DC CVAU](#), [DC CIVAC](#), [DC CVAC](#), [DC CVAP](#), and [IC IVAU](#).

If [FEAT_DPB2](#) is implemented, this trap also applies to [DC CVADP](#).

If [FEAT_MTE](#) is implemented, this trap also applies to [DC CIGVAC](#), [DC CIGDVAC](#), [DC CGVAC](#), [DC CGDVAC](#), [DC CGVAP](#), and [DC CGDVAP](#).

If [FEAT_DPB2](#) and [FEAT_MTE](#) are implemented, this trap also applies to [DC CGVADP](#) and [DC CGDVADP](#).

| UCI | Meaning |
|-----|--|
| 0b0 | Execution of the specified instructions at EL0 using AArch64 is trapped. |
| 0b1 | This control does not cause any instructions to be trapped. |

When [FEAT_VHE](#) is implemented, and the value of [HCR_EL2.{E2H, TGE}](#) is {1, 1}, this bit has no effect on execution at EL0.

If the Point of Coherency is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

EE, bit [25]

Endianness of data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime.

| EE | Meaning |
|-----|---|
| 0b0 | Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are little-endian. |
| 0b1 | Explicit data accesses at EL1, and stage 1 translation table walks in the EL1&0 translation regime are big-endian. |

If an implementation does not provide Big-endian support at Exception levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception levels higher than EL0, this bit is RES1.

The EE bit is permitted to be cached in a TLB.

When [FEAT_VHE](#) is implemented, and the value of [HCR_EL2.{E2H, TGE}](#) is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

E0E, bit [24]

Endianness of data accesses at EL0.

| E0E | Meaning |
|-----|--|
| 0b0 | Explicit data accesses at EL0 are little-endian. |
| 0b1 | Explicit data accesses at EL0 are big-endian. |

If an implementation only supports Little-endian accesses at EL0, then this bit is RES0. This option is not permitted when SCTLR_EL1.EE is RES1.

If an implementation only supports Big-endian accesses at EL0, then this bit is RES1. This option is not permitted when SCTLR_EL1.EE is RES0.

This bit has no effect on the endianness of LDTR, LDTRH, LDTRSH, LDTRSW, STTR, and STTRH instructions executed at EL1.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

SPAN, bit [23]

When FEAT_PAN is implemented:

Set Privileged Access Never, on taking an exception to EL1.

| SPAN | Meaning |
|------|--|
| 0b0 | PSTATE.PAN is set to 1 on taking an exception to EL1. |
| 0b1 | The value of PSTATE.PAN is left unchanged on taking an exception to EL1. |

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EIS, bit [22]

When FEAT_ExS is implemented:

Exception Entry is Context Synchronizing.

| EIS | Meaning |
|-----|---|
| 0b0 | The taking of an exception to EL1 is not a context synchronizing event. |
| 0b1 | The taking of an exception to EL1 is a context synchronizing event. |

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

If SCTLR_EL1.EIS is set to 0b0:

- Indirect writes to [ESR_EL1](#), [FAR_EL1](#), [SPSR_EL1](#), [ELR_EL1](#) are synchronized on exception entry to EL1, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.
- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL1.EIS:

- Changes to the PSTATE information on entry to EL1.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IESB, bit [21]

When FEAT_IESB is implemented:

Implicit Error Synchronization event enable. Possible values are:

| IESB | Meaning |
|------|--|
| 0b0 | Disabled. |
| 0b1 | An implicit error synchronization event is added: <ul style="list-style-type: none"> • At each exception taken to EL1. • Before the operational pseudocode of each ERET instruction executed at EL1. |

When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each DCPSX instruction taken to EL1 and before each DRPS instruction executed at EL1, in addition to the other cases where it is added.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TSCXT, bit [20]

When FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented:

Trap EL0 Access to the [SCXTNUM_EL0](#) register, when EL0 is using AArch64.

| TSCXT | Meaning |
|-------|--|
| 0b0 | EL0 access to SCXTNUM_EL0 is not disabled by this mechanism. |
| 0b1 | EL0 access to SCXTNUM_EL0 is disabled, causing an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1. The value of SCXTNUM_EL0 is treated as 0. |

When FEAT_VHE is implemented, and the value of [HCR_EL2.{E2H, TGE}](#) is {1,1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL1&0 translation regime, this bit can force all memory regions that are writable to be treated as XN.

| WXN | Meaning |
|-----|---|
| 0b0 | This control has no effect on memory access permissions. |
| 0b1 | Any region that is writable in the EL1&0 translation regime is forced to XN for accesses from software executing at EL1 or EL0. |

This bit applies only when SCTLR_EL1.M bit is set.

The WXN bit is permitted to be cached in a TLB.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

Traps EL0 execution of WFE instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2](#).TGE is 1, from both Execution states, reported using an ESR_ELx.EC value of 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFET instruction.

| nTWE | Meaning |
|------|---|
| 0b0 | Any attempt to execute a WFE instruction at EL0 is trapped, if the instruction would otherwise have caused the PE to enter a low-power state. |
| 0b1 | This control does not cause any instructions to be trapped. |

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

Traps EL0 execution of WFI instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2](#).TGE is 1, from both Execution states, reported using an ESR_ELx.EC value of 0x01.

When FEAT_WFxT is implemented, this trap also applies to the WFIT instruction.

| nTWI | Meaning |
|-------------|---|
| 0b0 | Any attempt to execute a WFI instruction at EL0 is trapped, if the instruction would otherwise have caused the PE to enter a low-power state. |
| 0b1 | This control does not cause any instructions to be trapped. |

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

UCT, bit [15]

Traps EL0 accesses to the [CTR_EL0](#) to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2](#).TGE is 1, from AArch64 state only, reported using an ESR_ELx.EC value of 0x18.

| UCT | Meaning |
|------------|---|
| 0b0 | Accesses to the CTR_EL0 from EL0 using AArch64 are trapped. |
| 0b1 | This control does not cause any instructions to be trapped. |

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

DZE, bit [14]

Traps EL0 execution of [DC ZVA](#) instructions to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2](#).TGE is 1, from AArch64 state only, reported using an ESR_ELx.EC value of 0x18.

If FEAT_MTE is implemented, this trap also applies to [DC GVA](#) and [DC GZVA](#).

| DZE | Meaning |
|------------|---|
| 0b0 | Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped. Reading DCZID_EL0 .DZP from EL0 returns 1, indicating that the instructions this trap applies to are not supported. |
| 0b1 | This control does not cause any instructions to be trapped. |

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

EnDB, bit [13]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication (using the APDBKey_EL1 key) of instruction addresses in the EL1&0 translation regime.

For more information, see 'System register control of pointer authentication'.

| EnDB | Meaning |
|------|--|
| 0b0 | Pointer authentication (using the APDBKey_EL1 key) of data addresses is not enabled. |
| 0b1 | Pointer authentication (using the APDBKey_EL1 key) of data addresses is enabled. |

Note

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. Specifically, when the field is 1, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Stage 1 instruction access Cacheability control, for accesses at EL0 and EL1:

| I | Meaning |
|-----|---|
| 0b0 | All instruction access to Stage 1 Normal memory from EL0 and EL1 are Stage 1 Non-cacheable. If the value of SCTLR_EL1.M is 0, instruction accesses from stage 1 of the EL1&0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory. |
| 0b1 | This control has no effect on the Stage 1 Cacheability of instruction access to Stage 1 Normal memory from EL0 and EL1. If the value of SCTLR_EL1.M is 0, instruction accesses from stage 1 of the EL1&0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory. |

When the value of the [HCR_EL2.DC](#) bit is 1, then instruction access to Normal memory from EL0 and EL1 are Cacheable regardless of the value of the SCTLR_EL1.I bit.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to 0.

EOS, bit [11]

When FEAT_ExS is implemented:

Exception Exit is Context Synchronizing.

| EOS | Meaning |
|------------|---|
| 0b0 | An exception return from EL1 is not a context synchronizing event |
| 0b1 | An exception return from EL1 is a context synchronizing event |

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

If SCTLR_EL1.EOS is set to 0b0:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL1.EOS:

- The indirect write of the PSTATE and PC values from [SPSR_EL1](#) and [ELR_EL1](#) on exception return is synchronized.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnRCTX, bit [10]

When FEAT_SPECRES is implemented:

Enable EL0 Access to the following instructions:

- AArch32 CFPRCTX, DVPRCTX and CPPRCTX instructions.
- AArch64 CFP RCTX, DVP RCT and CPP RCTX instructions.

| EnRCTX | Meaning |
|---------------|---|
| 0b0 | EL0 access to these instructions is disabled, and these instructions are trapped to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2 .TGE is 1. |
| 0b1 | EL0 access to these instructions is enabled. |

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1,1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UMA, bit [9]

User Mask Access. Traps EL0 execution of MSR and MRS instructions that access the PSTATE.{D, A, I, F} masks to EL1, or to EL2 when it is implemented and enabled for the current Security state and [HCR_EL2](#).TGE is 1, from AArch64 state only, reported using an ESR_ELx.EC value of 0x18.

| UMA | Meaning |
|-----|---|
| 0b0 | Any attempt at EL0 using AArch64 to execute an MRS, MSR(REGISTER), or MSR(IMMEDIATE) instruction that accesses the DAIF is trapped. |
| 0b1 | This control does not cause any instructions to be trapped. |

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

SED, bit [8]

When EL0 is capable of using AArch32:

SETEND instruction disable. Disables SETEND instructions at EL0 using AArch32.

| SED | Meaning |
|-----|---|
| 0b0 | SETEND instruction execution is enabled at EL0 using AArch32. |
| 0b1 | SETEND instructions are UNDEFINED at EL0 using AArch32 and any attempt at EL0 to access a SETEND instruction generates an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1, reported using an ESR_ELx.EC value of 0x00. |

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

When FEAT_VHE is implemented, and the value of HCR_EL2.{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

ITD, bit [7]

When EL0 is capable of using AArch32:

IT Disable. Disables some uses of IT instructions at EL0 using AArch32.

| ITD | Meaning |
|-----|--|
| 0b0 | All IT instruction functionality is enabled at EL0 using AArch32. |
| 0b1 | <p>Any attempt at EL0 using AArch32 to execute any of the following is UNDEFINED and generates an exception, reported using an ESR_ELx.EC value of 0x00, to EL1 or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2.TGE is 1:</p> <ul style="list-style-type: none"> • All encodings of the IT instruction with hw1[3:0] != 1000. • All encodings of the subsequent instruction with the following values for hw1: <ul style="list-style-type: none"> ◦ 0b11xxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM. ◦ 0b1011xxxxxxxxxxxx: All instructions in 'Miscellaneous 16-bit instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section F3.2.5. ◦ 0b10100xxxxxxxxxxx: ADD Rd, PC, #imm ◦ 0b01001xxxxxxxxxxx: LDR Rd, [PC, #imm] ◦ 0b0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC. ◦ 0b010001xx1xxxx111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers unpredictable cases with BLX Rn. <p>These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block.</p> <p>It is IMPLEMENTATION DEFINED whether the IT instruction is treated as:</p> <ul style="list-style-type: none"> • A 16-bit instruction, that can only be followed by another 16-bit instruction. • The first half of a 32-bit instruction. <p>This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED.</p> <p>An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.</p> |

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONSTRAINED UNPREDICTABLE. For more information, see 'Changes to an ITD control by an instruction in an IT block'.

ITD is optional, but if it is implemented in the SCTLR_EL1 then it must also be implemented in the [SCTLR_EL2](#), [HSTCLR](#), and [SCTLR](#).

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement ITD, access to this field is **RAZ/WI**.

Otherwise:

Reserved, RES1.

nAA, bit [6]

When FEAT_LSE2 is implemented:

Non-aligned access. This bit controls generation of Alignment faults at EL1 and EL0 under certain conditions.

| nAA | Meaning |
|-----|--|
| 0b0 | LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes for accesses. |
| 0b1 | This control bit does not cause LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, or STLURH to generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes. |

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CP15BEN, bit [5]

When EL0 is capable of using AArch32:

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from EL0:

| CP15BEN | Meaning |
|---------|--|
| 0b0 | EL0 using AArch32: EL0 execution of the CP15DMB , CP15DSB , and CP15ISB instructions is UNDEFINED and generates an exception to EL1, or to EL2 when it is implemented and enabled for the current Security state and HCR_EL2 .TGE is 1. The exception is reported using an ESR_ELx.EC value of 0x00. |
| 0b1 | EL0 using AArch32: EL0 execution of the CP15DMB , CP15DSB , and CP15ISB instructions is enabled. |

CP15BEN is optional, but if it is implemented in the SCTLR_EL1 then it must also be implemented in the [SCTLR_EL2](#), [HSCTLR](#), and [SCTLR](#).

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement CP15BEN, access to this field is **RAO/WI**.

Otherwise:

Reserved, RES0.

SA0, bit [4]

SP Alignment check enable for EL0. When set to 1, if a load or store instruction executed at EL0 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

SA, bit [3]

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL1 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Stage 1 Cacheability control, for data accesses.

| C | Meaning |
|-----|--|
| 0b0 | All data access to Stage 1 Normal memory from EL0 and EL1, and all Normal memory accesses from unified cache to the EL1&0 Stage 1 translation tables, are treated as Stage 1 Non-cacheable. |
| 0b1 | This control has no effect on the Stage 1 Cacheability of: <ul style="list-style-type: none"> Data access to Normal memory from EL0 and EL1. Normal memory accesses to the EL1&0 Stage 1 translation tables. |

When the value of the [HCR_EL2](#).DC bit is 1, the PE ignores SCTLR.C. This means that Non-secure EL0 and Non-secure EL1 data accesses to Normal memory are Cacheable.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to 0.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL1 and EL0.

| A | Meaning |
|-----|---|
| 0b0 | Alignment fault checking disabled when executing at EL1 or EL0. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed. |
| 0b1 | Alignment fault checking enabled when executing at EL1 or EL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception. |

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on execution at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL1&0 stage 1 address translation.

| M | Meaning |
|-----|---|
| 0b0 | EL1&0 stage 1 address translation disabled. See the SCTLR_EL1.I field for the behavior of instruction accesses to Normal memory. |
| 0b1 | EL1&0 stage 1 address translation enabled. |

If the value of [HCR_EL2](#).{DC, TGE} is not {0, 0} then in Non-secure state the PE behaves as if the value of the SCTLR_EL1.M field is 0 for all purposes other than returning the value of a direct read of the field.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this bit has no effect on the PE.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL1, this field resets to 0.

Accessing SCTLR_EL1

When [HCR_EL2](#).E2H is 1, without explicit synchronization, access from EL3 using the mnemonic SCTLR_EL1 or SCTLR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SCTLR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.SCTLR_EL1 == '1'
    then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x110];
    else
        return SCTLR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return SCTLR_EL2;
    else
        return SCTLR_EL1;
elsif PSTATE.EL == EL3 then
    return SCTLR_EL1;

```

MSR SCTLR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b000 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.SCTLR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x110] = X[t];
    else
        SCTLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        SCTLR_EL2 = X[t];
    else
        SCTLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    SCTLR_EL1 = X[t];

```

MRS <Xt>, SCTLR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x110];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return SCTLR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return SCTLR_EL1;
    else
        UNDEFINED;

```

MSR SCTLR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x110] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        SCTLR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        SCTLR_EL1 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLR_EL2, System Control Register (EL2)

The SCTLR_EL2 characteristics are:

Purpose

Provides top level control of the system, including its memory system, at EL2.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, these controls apply also to execution at EL0.

Configuration

AArch64 System register SCTLR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HSCTLR\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

SCTLR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | |
|-----------------------|---------------------------|------------------------|------------------------|----------------------|----------------------|-----------------------|-----------------------|-----------------------|---------------------|----------------------|-----------------------|----------------------|------------------------|----------------------|------------------------|---------------------|------------------------|---------------------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 |
| TIDCP | SPINTMASK | NMI | EnTP2 | RES0 | EPAN | EnALS | EnAS0 | EnASR | TME | TME0 | TMT | TMT0 | TWEDEL | | TWEDEL | | TWEDEL | |
| EnIA | EnIB | LSMAOE | nTLSMD | EnDA | UCI | EE | EOE | SPAN | EIS | IESB | TSCXT | WXN | nTWE | RES0 | nTWI | UCT | DZE | EnD |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 |

TIDCP, bit [63]

When FEAT_TIDCP1 is implemented and HCR_EL2.E2H == 1:

Trap IMPLEMENTATION DEFINED functionality. Traps EL0 accesses to the encodings reserved for IMPLEMENTATION DEFINED functionality to EL2.

| TIDCP | Meaning |
|-------|--|
| 0b0 | No instructions accessing the System register or System instruction spaces are trapped by this mechanism. |
| 0b1 | <p>If HCR_EL2.TGE==0, no instructions accessing the System register or System instruction spaces are trapped by this mechanism.</p> <p>If HCR_EL2.TGE==1, instructions accessing the following System register or System instruction spaces are trapped to EL2 by this mechanism:</p> <ul style="list-style-type: none"> In AArch64 state, EL0 access to the encodings in the following reserved encoding spaces are trapped and reported using EC syndrome 0x18: <ul style="list-style-type: none"> IMPLEMENTATION DEFINED System instructions, which are accessed using SYS and SYSL, with CRn == {11, 15}. IMPLEMENTATION DEFINED System registers, which are accessed using MRS and MSR with the S3_<op1>_<Cn>_<Cm>_<op2> register name. In AArch32 state, EL0 MCR and MRC access to the following encodings are trapped and reported using EC syndrome 0x03: <ul style="list-style-type: none"> All coproc==p15, CRn==c9, opc1 == {0-7}, CRm == {c0-c2, c5-c8}, opc2 == {0-7}. All coproc==p15, CRn==c10, opc1 == {0-7}, CRm == {c0, c1, c4, c8}, opc2 == {0-7}. All coproc==p15, CRn==c11, opc1 == {0-7}, CRm == {c0-c8, c15}, opc2 == {0-7}. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

Superpriority Interrupt Mask enable. When SCTLR_EL2.NMI is 1, controls the value of PSTATE.ALLINT on taking an exception to EL2.

| SPINTMASK | Meaning |
|-----------|--|
| 0b0 | PSTATE.ALLINT is set to 1 on taking an exception to EL2. |
| 0b1 | PSTATE.ALLINT is set to 0 on taking an exception to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMI, bit [61]

When FEAT_NMI is implemented:

Non-maskable Interrupt enable. Enables support for IRQ and FIQ interrupts with Superpriority, and determines additional masking behavior of the PSTATE.I and PSTATE.F flags.

| NMI | Meaning |
|-----|--|
| 0b0 | The behaviour of PSTATE.I and PSTATE.F is unchanged. IRQ and FIQ interrupts with Superpriority have no effect on interrupts that are targeted at EL2. |
| 0b1 | IRQ and FIQ interrupts can be marked as having Superpriority as an additional attribute, and additional Superpriority masking behavior is determined by PSTATE.ALLINT. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Otherwise:

Reserved, RES0.

EnTP2, bit [60]

When FEAT_SME is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Traps instructions executed at EL0 that access [TPIDR2_EL0](#) to EL2 when EL2 is implemented and enabled for the current Security state. The exception is reported using ESR_ELx.EC value 0x18.

| EnTP2 | Meaning |
|-------|---|
| 0b0 | This control causes execution of these instructions at EL0 to be trapped. |
| 0b1 | This control does not cause execution of any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [59:58]

Reserved, RES0.

EPAN, bit [57]

When FEAT_PAN3 is implemented and HCR_EL2.E2H == 1:

Enhanced Privileged Access Never. When PSTATE.PAN is 1, determines whether an EL2 data access to a page with EL0 instruction access permission generates a Permission fault as a result of the Privileged Access Never mechanism.

| EPAN | Meaning |
|------|---|
| 0b0 | No additional Permission faults are generated by this mechanism. |
| 0b1 | An EL2 data access to a page with stage 1 EL0 data access permission or stage 1 EL0 instruction access permission generates a Permission fault. Any speculative data accesses that would generate a Permission fault if the accesses were not speculative will not cause an allocation into a cache. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnALS, bit [56]

When FEAT_LS64 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Traps execution of an LD64B or ST64B instruction at EL0 to EL2.

| EnALS | Meaning |
|-------|--|
| 0b0 | Execution of an LD64B or ST64B instruction at EL0 is trapped to EL2. |
| 0b1 | This control does not cause any instructions to be trapped. |

A trap of an LD64B or ST64B instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000002.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnAS0, bit [55]

When FEAT_LS64 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Traps execution of an ST64BV0 instruction at EL0 to EL2.

| EnAS0 | Meaning |
|-------|---|
| 0b0 | Execution of an ST64BV0 instruction at EL0 is trapped to EL2. |
| 0b1 | This control does not cause any instructions to be trapped. |

A trap of an ST64BV0 instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000001.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnASR, bit [54]

When FEAT_LS64 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Traps execution of an ST64BV instruction at EL0 to EL2.

| EnASR | Meaning |
|-------|--|
| 0b0 | Execution of an ST64BV instruction at EL0 is trapped to EL2. |
| 0b1 | This control does not cause any instructions to be trapped. |

A trap of an ST64BV instruction is reported using an ESR_ELx.EC value of 0x0A, with an ISS code of 0x0000000.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME, bit [53]

When FEAT_TME is implemented:

Enables the Transactional Memory Extension at EL2.

| TME | Meaning |
|------------|---|
| 0b0 | Any attempt to execute a TSTART instruction at EL2 is trapped, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL2. |
| 0b1 | This control does not cause any TSTART instruction to be trapped. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TME0, bit [52]

When FEAT_TME is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Enables the Transactional Memory Extension at EL0.

| TME0 | Meaning |
|-------------|--|
| 0b0 | Any attempt to execute a TSTART instruction at EL0 is trapped to EL2, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL0. |
| 0b1 | This control does not cause any TSTART instruction to be trapped. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT, bit [51]

When FEAT_TME is implemented:

Forces a trivial implementation of the Transactional Memory Extension at EL2.

| TMT | Meaning |
|------------|---|
| 0b0 | This control does not cause any TSTART instruction to fail. |
| 0b1 | When the TSTART instruction is executed at EL2, the transaction fails with a TRIVIAL failure cause. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TMT0, bit [50]

When FEAT_TME is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Forces a trivial implementation of the Transactional Memory Extension at EL0.

| TMT0 | Meaning |
|-------------|---|
| 0b0 | This control does not cause any TSTART instruction to fail. |
| 0b1 | When the TSTART instruction is executed at EL0, the transaction fails with a TRIVIAL failure cause. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEL, bits [49:46]

When FEAT_TWED is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

TWE Delay. A 4-bit unsigned number that, when SCTLR_EL2.TWEDEn is 1, encodes the minimum delay in taking a trap of WFE caused by SCTLR_EL2.nTWE as $2^{(TWEDEL + 8)}$ cycles.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TWEDEn, bit [45]

When FEAT_TWED is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

TWE Delay Enable. Enables a configurable delayed trap of the WFE instruction caused by SCTLR_EL2.nTWE.

| TWEDEn | Meaning |
|---------------|---|
| 0b0 | The delay for taking a WFE trap is IMPLEMENTATION DEFINED. |
| 0b1 | The delay for taking a WFE trap is at least the number of cycles defined in SCTLR_EL2.TWEDEL. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DSSBS, bit [44]

When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry.

| DSSBS | Meaning |
|-------|---|
| 0b0 | PSTATE.SSBS is set to 0 on an exception to EL2. |
| 0b1 | PSTATE.SSBS is set to 1 on an exception to EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When FEAT_MTE2 is implemented:

Allocation Tag Access in EL2. When [SCR_EL3.ATA](#) is 1, controls EL2 access to Allocation Tags.

| ATA | Meaning |
|-----|--|
| 0b0 | Access to Allocation Tags is prevented. |
| 0b1 | This control does not prevent access to Allocation Tags. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ATA0, bit [42]

When FEAT_MTE2 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Allocation Tag Access in EL0. When [SCR_EL3.ATA](#) is 1, controls EL0 access to Allocation Tags.

| ATA0 | Meaning |
|------|--|
| 0b0 | Access to Allocation Tags is prevented. |
| 0b1 | This control does not prevent access to Allocation Tags. |

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF, bits [41:40]

When FEAT_MTE2 is implemented:

Tag Check Fault in EL2. Controls the effect of Tag Check Faults due to Loads and Stores in EL2.

| TCF | Meaning | Applies when |
|------|--|-------------------------------|
| 0b00 | Tag Check Faults have no effect on the PE. | When FEAT_MTE3 is implemented |
| 0b01 | Tag Check Faults cause a synchronous exception. | |
| 0b10 | Tag Check Faults are asynchronously accumulated. | |
| 0b11 | Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes. | |

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCF0, bits [39:38]

When FEAT_MTE2 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Tag Check Fault in EL0. Controls the effect of Tag Check Faults due to Loads and Stores in EL0.

| TCF0 | Meaning | Applies when |
|------|--|-------------------------------|
| 0b00 | Tag Check Faults have no effect on the PE. | When FEAT_MTE3 is implemented |
| 0b01 | Tag Check Faults cause a synchronous exception. | |
| 0b10 | Tag Check Faults are asynchronously accumulated. | |
| 0b11 | Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes. | |

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ITFSB, bit [37]**When FEAT_MTE2 is implemented:**

When synchronous exceptions are not being generated by Tag Check Faults, this field controls whether on exception entry into EL2, all Tag Check Faults due to instructions executed before exception entry, that are reported asynchronously, are synchronized into [TFSRE0_EL1](#), [TFSR_EL1](#) and [TFSR_EL2](#) registers.

| ITFSB | Meaning |
|-------|--|
| 0b0 | Tag Check Faults are not synchronized on entry to EL2. |
| 0b1 | Tag Check Faults are synchronized on entry to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT, bit [36]**When FEAT_BT1 is implemented:**

PAC Branch Type compatibility at EL2.

When [HCR_EL2](#).{E2H, TGE} == {1, 1}, this bit is named BT1.

| BT | Meaning |
|-----|--|
| 0b0 | When the PE is executing at EL2, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11. |
| 0b1 | When the PE is executing at EL2, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT0, bit [35]**When FEAT_BT1 is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:**

PAC Branch Type compatibility at EL0.

| BT0 | Meaning |
|-----|--|
| 0b0 | When the PE is executing at EL0, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11. |
| 0b1 | When the PE is executing at EL0, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [34]

Reserved, RES0.

MSCEn, bit [33]

When FEAT_MOPS is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

MemCpy and MemSet instructions Enable. Enables execution of the MemCpy and MemSet instructions at EL0.

| MSCEn | Meaning |
|-------|--|
| 0b0 | Execution of the MemCpy and MemSet instructions is UNDEFINED at EL0. |
| 0b1 | This control does not cause any instructions to be UNDEFINED. |

When FEAT_MOPS is implemented and [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the Effective value of this bit is 0b1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CMOW, bit [32]

When FEAT_CMOW is implemented and HCR_EL2.E2H == 1:

Controls cache maintenance instruction permission for the following instructions executed at EL0.

- [IC IVAU](#), [DC CIVAC](#), [DC CIGDVAC](#) and [DC CIGVAC](#).

| CMOW | Meaning |
|------|--|
| 0b0 | These instructions executed at EL0 with stage 1 read permission, but without stage 1 write permission, do not generate a stage 1 permission fault. |
| 0b1 | If enabled as a result of SCTLR_EL2 .UCI==1, these instructions executed at EL0 with stage 1 read permission, but without stage 1 write permission, generate a stage 1 permission fault. |

When [HCR_EL2](#).TGE is 0, this bit has no effect on execution at EL0.

For this control, stage 1 has write permission if all of the following apply:

- AP[2] is 0 or DBM is 1 in the stage 1 descriptor.
- Where APTTable is in use, APTTable[1] is 0 for all levels of the translation table.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIA, bit [31]**When FEAT_PAuth is implemented:**

Controls enabling of pointer authentication (using the APIAKey_EL1 key) of instruction addresses in the EL2 or EL2&0 translation regime.

For more information, see 'System register control of pointer authentication'.

| EnIA | Meaning |
|------|---|
| 0b0 | Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is not enabled. |
| 0b1 | Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is enabled. |

Note

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. Specifically, when the field is 1, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]**When FEAT_PAuth is implemented:**

Controls enabling of pointer authentication (using the APIBKey_EL1 key) of instruction addresses in the EL2 or EL2&0 translation regime.

For more information, see 'System register control of pointer authentication'.

| EnIB | Meaning |
|------|---|
| 0b0 | Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is not enabled. |
| 0b1 | Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is enabled. |

Note

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. Specifically, when the field is 1, AddPACIB returns a copy of a pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

LSMAOE, bit [29]

When FEAT_LSMAOC is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Load Multiple and Store Multiple Atomicity and Ordering Enable.

| LSMAOE | Meaning |
|--------|---|
| 0b0 | For all memory accesses at EL0, A32 and T32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered. |
| 0b1 | The ordering and interrupt behavior of A32 and T32 Load Multiple and Store Multiple at EL0 is as defined for Armv8.0. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

nTLSMD, bit [28]

When FEAT_LSMAOC is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

| nTLSMD | Meaning |
|--------|--|
| 0b0 | All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault. |
| 0b1 | All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnDA, bit [27]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication (using the APDAKey_EL1 key) of instruction addresses in the EL2 or EL2&0 translation regime.

For more information, see 'System register control of pointer authentication'.

| EnDA | Meaning |
|------|--|
| 0b0 | Pointer authentication (using the APDAKey_EL1 key) of data addresses is not enabled. |
| 0b1 | Pointer authentication (using the APDAKey_EL1 key) of data addresses is enabled. |

Note

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. Specifically, when the field is 1, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UCI, bit [26]

When `HCR_EL2.E2H == 1` and `HCR_EL2.TGE == 1`:

Traps execution of cache maintenance instructions at EL0 to EL2, from AArch64 state only. This applies to [DC CVAU](#), [DC CIVAC](#), [DC CVAC](#), [DC CVAP](#), and [IC IVAU](#).

If FEAT_DPB2 is implemented, this trap also applies to [DC CVADP](#).

If FEAT_MTE is implemented, this trap also applies to [DC CIGVAC](#), [DC CIGDVAC](#), [DC CGVAC](#), [DC CGDVAC](#), [DC CGVAP](#), and [DC CGDVAP](#).

If FEAT_DPB2 and FEAT_MTE are implemented, this trap also applies to [DC CGVADP](#) and [DC CGDVADP](#).

| UCI | Meaning |
|-----|---|
| 0b0 | Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped to EL2. |
| 0b1 | This control does not cause any instructions to be trapped. |

If the Point of Coherency is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EE, bit [25]

Endianness of data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime.

| EE | Meaning |
|-----|---|
| 0b0 | Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime are little-endian. |
| 0b1 | Explicit data accesses at EL2, stage 1 translation table walks in the EL2 or EL2&0 translation regime, and stage 2 translation table walks in the EL1&0 translation regime are big-endian. |

If an implementation does not provide Big-endian support at Exception levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception levels higher than EL0, this bit is RES1.

The EE bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

EOE, bit [24]

When HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Endianness of data accesses at EL0.

| EOE | Meaning |
|-----|--|
| 0b0 | Explicit data accesses at EL0 are little-endian. |
| 0b1 | Explicit data accesses at EL0 are big-endian. |

If an implementation only supports Little-endian accesses at EL0, then this bit is RES0. This option is not permitted when SCTLR_EL1.EE is RES1.

If an implementation only supports Big-endian accesses at EL0, then this bit is RES1. This option is not permitted when SCTLR_EL1.EE is RES0.

This bit has no effect on the endianness of LDTR, LDTRH, LDTRSH, LDTRSW, STTR, and STTRH instructions executed at EL1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SPAN, bit [23]

When HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Set Privileged Access Never, on taking an exception to EL2.

| SPAN | Meaning |
|------|--|
| 0b0 | PSTATE.PAN is set to 1 on taking an exception to EL2. |
| 0b1 | The value of PSTATE.PAN is left unchanged on taking an exception to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EIS, bit [22]**When FEAT_ExS is implemented:**

Exception entry is a context synchronization event.

| EIS | Meaning |
|------------|---|
| 0b0 | The taking of an exception to EL2 is not a context synchronization event. |
| 0b1 | The taking of an exception to EL2 is a context synchronization event. |

If SCTLR_EL2.EIS is set to 0b0:

- Indirect writes to [ESR_EL2](#), [FAR_EL2](#), [SPSR_EL2](#), [ELR_EL2](#), and [HPFAR_EL2](#) are synchronized on exception entry to EL2, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.
- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL2.EIS:

- Changes to the PSTATE information on entry to EL2.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores, and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IESB, bit [21]**When FEAT_IESB is implemented:**

Implicit Error Synchronization event enable.

| IESB | Meaning |
|-------------|--|
| 0b0 | Disabled. |
| 0b1 | An implicit error synchronization event is added: <ul style="list-style-type: none"> • At each exception taken to EL2. • Before the operational pseudocode of each ERET instruction executed at EL2. |

When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field. If the Effective value of the field is 1, then an implicit error synchronization event is added after each DCPSX instruction taken to EL2 and before each DRPS instruction executed at EL2, in addition to the other cases where it is added.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TSCXT, bit [20]

When (FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented), HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Trap EL0 Access to the SCXTNUM_EL0 register, when EL0 is using AArch64.

| TSCXT | Meaning |
|--------------|--|
| 0b0 | EL0 access to SCXTNUM_EL0 is not disabled by this mechanism. |
| 0b1 | EL0 access to SCXTNUM_EL0 is disabled, causing an exception to EL2, and the SCXTNUM_EL0 value is treated as 0. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

When FEAT_CSV2_2 is not implemented, FEAT_CSV2_1p2 is not implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Reserved, RES1.

Otherwise:

Reserved, RES0.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL2 or EL2&0 translation regime, this bit can force all memory regions that are writable to be treated as XN.

| WXN | Meaning |
|------------|---|
| 0b0 | This control has no effect on memory access permissions. |
| 0b1 | Any region that is writable in the EL2 or EL2&0 translation regime is forced to XN for accesses from software executing at EL2. |

This bit applies only when SCTLR_EL2.M bit is set.

The WXN bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

nTWE, bit [18]

When HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Traps execution of WFE instructions at EL0 to EL2, from both Execution states.

| nTWE | Meaning |
|-------------|--|
| 0b0 | Any attempt to execute a WFE instruction at EL0 is trapped to EL2, if the instruction would otherwise have caused the PE to enter a low-power state. |
| 0b1 | This control does not cause any instructions to be trapped. |

In AArch32 state, the attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

When HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Traps execution of WFI instructions at EL0 to EL2, from both Execution states.

| nTWI | Meaning |
|------|---|
| 0b0 | Any attempt to execute a WFI instruction at EL0 is trapped EL2, if the instruction would otherwise have caused the PE to enter a low-power state. |
| 0b1 | This control does not cause any instructions to be trapped. |

In AArch32 state, the attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

UCT, bit [15]

When HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Traps EL0 accesses to the [CTR_ELO](#) to EL2, from AArch64 state only.

| UCT | Meaning |
|-----|--|
| 0b0 | Accesses to the CTR_ELO from EL0 using AArch64 are trapped to EL2. |
| 0b1 | This control does not cause any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DZE, bit [14]

When HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Traps execution of [DC ZVA](#) instructions at EL0 to EL2, from AArch64 state only.

If FEAT_MTE is implemented, this trap also applies to [DC GVA](#) and [DC GZVA](#).

| DZE | Meaning |
|-----|---|
| 0b0 | Any attempt to execute an instruction that this trap applies to at EL0 using AArch64 is trapped to EL2. Reading DCZID_EL0.DZP from EL0 returns 1, indicating that the instructions that this trap applies to are not supported. |
| 0b1 | This control does not cause any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnDB, bit [13]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication (using the APDBKey_EL1 key) of instruction addresses in the EL2 or EL2&0 translation regime.

For more information, see 'System register control of pointer authentication'.

| EnDB | Meaning |
|------|--|
| 0b0 | Pointer authentication (using the APDBKey_EL1 key) of data addresses is not enabled. |
| 0b1 | Pointer authentication (using the APDBKey_EL1 key) of data addresses is enabled. |

Note

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. Specifically, when the field is 1, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Instruction access Cacheability control, for accesses at EL2 and, when EL2 is enabled in the current Security state and [HCR_EL2](#).{E2H,TGE} == {1,1}, EL0.

| I | Meaning |
|----------|---|
| 0b0 | All instruction accesses to Normal memory from EL2 are Non-cacheable for all levels of instruction and unified cache. When EL2 is enabled in the current Security state and HCR_EL2 .{E2H, TGE} == {1, 1}, all instruction accesses to Normal memory from EL0 are Non-cacheable for all levels of instruction and unified cache. If SCTLR_EL2.M is 0, instruction accesses from stage 1 of the EL2 or EL2&0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory. |
| 0b1 | This control has no effect on the Cacheability of instruction access to Normal memory from EL2 and, when EL2 is enabled in the current Security state and HCR_EL2 .{E2H, TGE} == {1, 1}, instruction access to Normal memory from EL0. If the value of SCTLR_EL2.M is 0, instruction accesses from stage 1 of the EL2 or EL2&0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory. |

This bit has no effect on the EL3 translation regime.

When EL2 is disabled in the current Security state or [HCR_EL2](#).{E2H,TGE} != {1,1}, this bit has no effect on the EL1&0 translation regime.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

EOS, bit [11]

When FEAT_ExS is implemented:

Exception exit is a context synchronization event.

| EOS | Meaning |
|------------|--|
| 0b0 | An exception return from EL2 is not a context synchronization event. |
| 0b1 | An exception return from EL2 is a context synchronization event. |

If SCTLR_EL2.EOS is set to 0b0:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL2.EOS:

- The indirect write of the PSTATE and PC values from [SPSR_EL2](#) and [ELR_EL2](#) on exception return is synchronized.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores, and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

EnRCTX, bit [10]

When FEAT_SPECRES is implemented, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Enable EL0 Access to the following instructions:

- AArch32 CFPRCTX, DVPRCTX and CPPRCTX instructions.
- AArch64 CFP RCTX, DVP RCT and CPP RCTX instructions.

| EnRCTX | Meaning |
|--------|--|
| 0b0 | EL0 access to these instructions is disabled, and these instructions are trapped to EL1. |
| 0b1 | EL0 access to these instructions is enabled. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [9]

Reserved, RES0.

SED, bit [8]

When EL0 is capable of using AArch32, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

SETEND instruction disable. Disables SETEND instructions at EL0 using AArch32.

| SED | Meaning |
|-----|---|
| 0b0 | SETEND instruction execution is enabled at EL0 using AArch32. |
| 0b1 | SETEND instructions are UNDEFINED at EL0 using AArch32. |

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

When EL0 can only use AArch64, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Reserved, RES1.

Otherwise:

Reserved, RES0.

ITD, bit [7]

When EL0 is capable of using AArch32, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

IT Disable. Disables some uses of IT instructions at EL0 using AArch32.

| ITD | Meaning |
|-----|---|
| 0b0 | All IT instruction functionality is enabled at EL0 using AArch32. |
| 0b1 | Any attempt at EL0 using AArch32 to execute any of the following is UNDEFINED: <ul style="list-style-type: none"> • All encodings of the IT instruction with hw1[3:0] != 1000. • All encodings of the subsequent instruction with the following values for hw1: <ul style="list-style-type: none"> ◦ 0b11xxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM. ◦ 0b1011xxxxxxxxxxxx: All instructions in 'Miscellaneous 16-bit instructions' in the Arm® Architecture Reference Manual, Armv8, for Armv8-A architecture profile, section F3.2.5. ◦ 0b10100xxxxxxxxxxx: ADD Rd, PC, #imm ◦ 0b01001xxxxxxxxxxx: LDR Rd, [PC, #imm] ◦ 0b0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC. ◦ 0b010001xx1xxxx111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers UNPREDICTABLE cases with BLX Rn. <p>These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block.</p> <p>It is IMPLEMENTATION DEFINED whether the IT instruction is treated as:</p> <ul style="list-style-type: none"> • A 16-bit instruction, that can only be followed by another 16-bit instruction. • The first half of a 32-bit instruction. <p>This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED.</p> <p>An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.</p> |

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONSTRAINED UNPREDICTABLE. For more information see 'Changes to an ITD control by an instruction in an IT block'.

ITD is optional, but if it is implemented in the SCTLR_EL2 then it must also be implemented in the [SCTLR_EL1](#), [HSCTLR](#), and [SCTLR](#).

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement ITD, access to this field is **RAZ/WI**.

When EL0 can only use AArch64, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Reserved, RES1.

Otherwise:

Reserved, RES0.

nAA, bit [6]

When FEAT_LSE2 is implemented:

Non-aligned access. This bit controls generation of Alignment faults under certain conditions at EL2, and, when EL2 is enabled in the current Security state and [HCR_EL2](#).{E2H, TGE} == {1, 1}, EL0.

| nAA | Meaning |
|-----|--|
| 0b0 | LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes for accesses. |
| 0b1 | This control bit does not cause LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, or STLURH to generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CP15BEN, bit [5]

When EL0 is capable of using AArch32, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from EL0:

| CP15BEN | Meaning |
|---------|--|
| 0b0 | EL0 using AArch32: EL0 execution of the CP15DMB , CP15DSB , and CP15ISB instructions is UNDEFINED. |
| 0b1 | EL0 using AArch32: EL0 execution of the CP15DMB , CP15DSB , and CP15ISB instructions is enabled. |

CP15BEN is optional, but if it is implemented in the SCTLR_EL2 then it must also be implemented in the [SCTLR_EL1](#), [HSCTLR](#), and [SCTLR](#).

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement CP15BEN, access to this field is **RAO/WI**.

When EL0 can only use AArch64, HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

Reserved, RES0.

Otherwise:

Reserved, RES1.

SA0, bit [4]

When HCR_EL2.E2H == 1 and HCR_EL2.TGE == 1:

SP Alignment check enable for EL0. When set to 1, if a load or store instruction executed at EL0 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

SA, bit [3]

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL2 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then an SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Data access Cacheability control, for accesses at EL2 and, when EL2 is enabled in the current Security state and [HCR_EL2](#).{E2H, TGE} == {1, 1}, EL0

| C | Meaning |
|-----|---|
| 0b0 | <p>The following are Non-cacheable for all levels of data and unified cache:</p> <ul style="list-style-type: none"> Data accesses to Normal memory from EL2. When HCR_EL2.{E2H, TGE} != {1, 1}, Normal memory accesses to the EL2 translation tables. When EL2 is enabled in the current Security state and HCR_EL2.{E2H, TGE} == {1, 1}: <ul style="list-style-type: none"> Data accesses to Normal memory from EL0. Normal memory accesses to the EL2&0 translation tables. |
| 0b1 | <p>This control has no effect on the Cacheability of:</p> <ul style="list-style-type: none"> Data access to Normal memory from EL2. When HCR_EL2.{E2H, TGE} != {1, 1}, Normal memory accesses to the EL2 translation tables. When EL2 is enabled in the current Security state and HCR_EL2.{E2H, TGE} == {1, 1}: <ul style="list-style-type: none"> Data accesses to Normal memory from EL0. Normal memory accesses to the EL2&0 translation tables. |

This bit has no effect on the EL3 translation regime.

When EL2 is disabled in the current Security state or [HCR_EL2](#).{E2H, TGE} != {1, 1}, this bit has no effect on the EL1&0 translation regime.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL2 and, when EL2 is enabled in the current Security state and [HCR_EL2](#).{E2H, TGE} == {1, 1}, EL0.

| A | Meaning |
|-----|--|
| 0b0 | Alignment fault checking disabled when executing at EL2. When EL2 is enabled in the current Security state and HCR_EL2 .{E2H, TGE} == {1, 1}, alignment fault checking disabled when executing at EL0. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed. |
| 0b1 | Alignment fault checking enabled when executing at EL2. When EL2 is enabled in the current Security state and HCR_EL2 .{E2H, TGE} == {1, 1}, alignment fault checking enabled when executing at EL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception. |

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL2 or EL2&0 stage 1 address translation.

| M | Meaning |
|-----|--|
| 0b0 | When HCR_EL2 .{E2H, TGE} != {1, 1}, EL2 stage 1 address translation disabled. When HCR_EL2 .{E2H, TGE} == {1, 1}, EL2&0 stage 1 address translation disabled. See the SCTLR_EL2.I field for the behavior of instruction accesses to Normal memory. |
| 0b1 | When HCR_EL2 .{E2H, TGE} != {1, 1}, EL2 stage 1 address translation enabled. When HCR_EL2 .{E2H, TGE} == {1, 1}, EL2&0 stage 1 address translation enabled. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Accessing SCTLR_EL2

When [HCR_EL2](#).E2H is 1, without explicit synchronization, access from EL2 using the mnemonic SCTLR_EL2 or SCTLR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SCTLR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return SCTLR_EL2;
elsif PSTATE.EL == EL3 then
    return SCTLR_EL2;

```

MSR SCTLR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    SCTLR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    SCTLR_EL2 = X[t];

```

MRS <Xt>, SCTLR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.SCTLR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x110];
    else
        return SCTLR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return SCTLR_EL2;
    else
        return SCTLR_EL1;
elsif PSTATE.EL == EL3 then
    return SCTLR_EL1;

```

MSR SCTLR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.SCTLR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x110] = X[t];
    else
        SCTLR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        SCTLR_EL2 = X[t];
    else
        SCTLR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    SCTLR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLR_EL3, System Control Register (EL3)

The SCTLR_EL3 characteristics are:

Purpose

Provides top level control of the system, including its memory system, at EL3.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to SCTLR_EL3 are UNDEFINED.

Attributes

SCTLR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----------|------|------|------|------|------|------|-----|------|------|------|------|------|------|------|------|----|----|-------|-----|------|-----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 |
| RES0 | SPINTMASK | NMI | | | RES0 | | | | | TME | RES0 | TMT | | | RES0 | | | | DSSBS | ATA | RES0 | TCF | | |
| EnIA | EnIB | RES1 | EnDA | RES0 | EE | RES0 | RES1 | EIS | IESB | RES0 | WXN | RES1 | RES0 | RES1 | RES0 | EnDB | | I | EOS | | RES0 | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 |

Bit [63]

Reserved, RES0.

SPINTMASK, bit [62]

When FEAT_NMI is implemented:

Superpriority Interrupt Mask enable. When SCTLR_EL3.NMI is 1, controls the value of PSTATE.ALLINT on taking an exception to EL3.

| SPINTMASK | Meaning |
|-----------|--|
| 0b0 | PSTATE.ALLINT is set to 1 on taking an exception to EL3. |
| 0b1 | PSTATE.ALLINT is set to 0 on taking an exception to EL3. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NMI, bit [61]

When FEAT_NMI is implemented:

Non-maskable Interrupt enable. Enables support for IRQ and FIQ interrupts with Superpriority, and determines additional masking behavior of the PSTATE.I and PSTATE.F flags.

| NMI | Meaning |
|-----|--|
| 0b0 | The behaviour of PSTATE.I and PSTATE.F is unchanged. IRQ and FIQ interrupts with Superpriority have no effect on interrupts that are targeted at EL3. |
| 0b1 | IRQ and FIQ interrupts can be marked as having Superpriority as an additional attribute, and additional Superpriority masking behavior is determined by PSTATE.ALLINT. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [60:54]

Reserved, RES0.

TME, bit [53]

When FEAT_TME is implemented:

Enables the Transactional Memory Extension at EL3.

| TME | Meaning |
|-----|---|
| 0b0 | Any attempt to execute a TSTART instruction at EL3 is trapped, unless HCR_EL2.TME or SCR_EL3.TME causes TSTART instructions to be UNDEFINED at EL3. |
| 0b1 | This control does not cause any TSTART instruction to be trapped. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [52]

Reserved, RES0.

TMT, bit [51]

When FEAT_TME is implemented:

Forces a trivial implementation of the Transactional Memory Extension at EL3.

| TMT | Meaning |
|-----|---|
| 0b0 | This control does not cause any TSTART instruction to fail. |
| 0b1 | When the TSTART instruction is executed at EL3, the transaction fails with a TRIVIAL failure cause. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [50:45]

Reserved, RES0.

DSSBS, bit [44]

When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry.

| DSSBS | Meaning |
|-------|---|
| 0b0 | PSTATE.SSBS is set to 0 on an exception to EL3. |
| 0b1 | PSTATE.SSBS is set to 1 on an exception to EL3. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

ATA, bit [43]

When FEAT_MTE2 is implemented:

Allocation Tag Access in EL3. Controls EL3 access to Allocation Tags.

| ATA | Meaning |
|-----|--|
| 0b0 | Access to Allocation Tags is prevented. |
| 0b1 | This control does not prevent access to Allocation Tags. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [42]

Reserved, RES0.

TCF, bits [41:40]

When FEAT_MTE2 is implemented:

Tag Check Fault in EL3. Controls the effect of Tag Check Faults due to Loads and Stores in EL3.

If FEAT_MTE3 is not implemented, the value 0b11 is reserved.

| TCF | Meaning | Applies when |
|------|--|-------------------------------|
| 0b00 | Tag Check Faults have no effect on the PE. | |
| 0b01 | Tag Check Faults cause a synchronous exception. | |
| 0b10 | Tag Check Faults are asynchronously accumulated. | |
| 0b11 | Tag Check Faults cause a synchronous exception on reads, and are asynchronously accumulated on writes. | When FEAT_MTE3 is implemented |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [39:38]

Reserved, RES0.

ITFSB, bit [37]

When FEAT_MTE2 is implemented:

When synchronous exceptions are not being generated by Tag Check Faults, this field controls whether on exception entry into EL3, all Tag Check Faults due to instructions executed before exception entry, that are reported asynchronously, are synchronized into [TFSRE0_EL1](#) and TFSR_ELx registers.

| ITFSB | Meaning |
|-------|--|
| 0b0 | Tag Check Faults are not synchronized on entry to EL3. |
| 0b1 | Tag Check Faults are synchronized on entry to EL3. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BT, bit [36]

When FEAT_BTI is implemented:

PAC Branch Type compatibility at EL3.

| BT | Meaning |
|-----|--|
| 0b0 | When the PE is executing at EL3, PACIASP and PACIBSP are compatible with PSTATE.BTYPE == 0b11. |
| 0b1 | When the PE is executing at EL3, PACIASP and PACIBSP are not compatible with PSTATE.BTYPE == 0b11. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [35:32]

Reserved, RES0.

EnIA, bit [31]**When FEAT_PAuth is implemented:**

Controls enabling of pointer authentication (using the APIAKey_EL1 key) of instruction addresses in the EL3 translation regime.

Possible values of this bit are:

| EnIA | Meaning |
|------|---|
| 0b0 | Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is not enabled. |
| 0b1 | Pointer authentication (using the APIAKey_EL1 key) of instruction addresses is enabled. |

For more information, see 'System register control of pointer authentication'.

Note

This field controls the behavior of the AddPACIA and AuthIA pseudocode functions. Specifically, when the field is 1, AddPACIA returns a copy of a pointer to which a pointer authentication code has been added, and AuthIA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EnIB, bit [30]**When FEAT_PAuth is implemented:**

Controls enabling of pointer authentication (using the APIBKey_EL1 key) of instruction addresses in the EL3 translation regime.

Possible values of this bit are:

| EnIB | Meaning |
|------|---|
| 0b0 | Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is not enabled. |
| 0b1 | Pointer authentication (using the APIBKey_EL1 key) of instruction addresses is enabled. |

For more information, see 'System register control of pointer authentication'.

Note

This field controls the behavior of the AddPACIB and AuthIB pseudocode functions. Specifically, when the field is 1, AddPACIB returns a copy of a

pointer to which a pointer authentication code has been added, and AuthIB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:28]

Reserved, RES1.

EnDA, bit [27]

When FEAT_PAuth is implemented:

Controls enabling of pointer authentication (using the APDAKey_EL1 key) of instruction addresses in the EL3 translation regime.

| EnDA | Meaning |
|------|--|
| 0b0 | Pointer authentication (using the APDAKey_EL1 key) of data addresses is not enabled. |
| 0b1 | Pointer authentication (using the APDAKey_EL1 key) of data addresses is enabled. |

For more information, see 'System register control of pointer authentication'.

Note

This field controls the behavior of the AddPACDA and AuthDA pseudocode functions. Specifically, when the field is 1, AddPACDA returns a copy of a pointer to which a pointer authentication code has been added, and AuthDA returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [26]

Reserved, RES0.

EE, bit [25]

Endianness of data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime.

| EE | Meaning |
|-----|---|
| 0b0 | Explicit data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime are little-endian. |
| 0b1 | Explicit data accesses at EL3, and stage 1 translation table walks in the EL3 translation regime are big-endian. |

If an implementation does not provide Big-endian support at Exception levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception levels higher than EL0, this bit is RES1.

The EE bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Bit [24]

Reserved, RES0.

Bit [23]

Reserved, RES1.

EIS, bit [22]

When FEAT_ExS is implemented:

Exception Entry is Context Synchronizing.

| EIS | Meaning |
|-----|---|
| 0b0 | The taking of an exception to EL3 is not a context synchronizing event. |
| 0b1 | The taking of an exception to EL3 is a context synchronizing event. |

If SCTLR_EL3.EIS is set to 0b0:

- Indirect writes to [ESR_EL3](#), [FAR_EL3](#), [SPSR_EL3](#), [ELR_EL3](#) are synchronized on exception entry to EL3, so that a direct read of the register after exception entry sees the indirectly written value caused by the exception entry.
- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL3.EIS:

- Changes to the PSTATE information on entry to EL3.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Debug state exit.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

IESB, bit [21]

When FEAT_IESB is implemented:

Implicit Error Synchronization event enable.

| IESB | Meaning |
|-------------|--|
| 0b0 | Disabled. |
| 0b1 | An implicit error synchronization event is added: <ul style="list-style-type: none"> • At each exception taken to EL3. • Before the operational pseudocode of each ERET instruction executed at EL3. |

When the PE is in Debug state, the effect of this field is CONSTRAINED UNPREDICTABLE, and its Effective value might be 0 or 1 regardless of the value of the field and, if implemented, [SCR_EL3.NMEA](#). If the Effective value of the field is 1, then an implicit error synchronization event is added after each DCPSX instruction taken to EL3 and before each DRPS instruction executed at EL3, in addition to the other cases where it is added.

When FEAT_DoubleFault is implemented, the PE is in Non-debug state, and the Effective value of [SCR_EL3.NMEA](#) is 1, this field is ignored and its Effective value is 1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [20]

Reserved, RES0.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL3 translation regime, this bit can force all memory regions that are writable to be treated as XN.

| WXN | Meaning |
|------------|--|
| 0b0 | This control has no effect on memory access permissions. |
| 0b1 | Any region that is writable in the EL3 translation regime is forced to XN for accesses from software executing at EL3. |

This bit applies only when SCTLR_EL3.M bit is set.

The WXN bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Bit [18]

Reserved, RES1.

Bit [17]

Reserved, RES0.

Bit [16]

Reserved, RES1.

Bits [15:14]

Reserved, RES0.

EnDB, bit [13]**When FEAT_PAuth is implemented:**

Controls enabling of pointer authentication (using the APDBKey_EL1 key) of instruction addresses in the EL3 translation regime.

| EnDB | Meaning |
|------|--|
| 0b0 | Pointer authentication (using the APDBKey_EL1 key) of data addresses is not enabled. |
| 0b1 | Pointer authentication (using the APDBKey_EL1 key) of data addresses is enabled. |

For more information, see 'System register control of pointer authentication'.

Note

This field controls the behavior of the AddPACDB and AuthDB pseudocode functions. Specifically, when the field is 1, AddPACDB returns a copy of a pointer to which a pointer authentication code has been added, and AuthDB returns an authenticated copy of a pointer. When the field is 0, both of these functions are NOP.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

I, bit [12]

Instruction access Cacheability control, for accesses at EL3:

| I | Meaning |
|-----|---|
| 0b0 | All instruction access to Normal memory from EL3 are Non-cacheable for all levels of instruction and unified cache. If the value of SCTLR_EL3.M is 0, instruction accesses from stage 1 of the EL3 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory. |
| 0b1 | This control has no effect on the Cacheability of instruction access to Normal memory from EL3. If the value of SCTLR_EL3.M is 0, instruction accesses from stage 1 of the EL3 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory. |

This bit has no effect on the EL1&0, EL2, or EL2&0 translation regimes.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

EOS, bit [11]**When FEAT_ExS is implemented:**

Exception Exit is Context Synchronizing.

| EOS | Meaning |
|-----|---|
| 0b0 | An exception return from EL3 is not a context synchronizing event |
| 0b1 | An exception return from EL3 is a context synchronizing event |

If SCTLR_EL3.EOS is set to 0b0:

- Memory transactions, including instruction fetches, from an Exception level always use the translation resources associated with that translation regime.
- Exception Catch debug events are synchronous debug events.
- DCPS* and DRPS instructions are context synchronization events.

The following are not affected by the value of SCTLR_EL3.EOS:

- The indirect write of the PSTATE and PC values from [SPSR_EL3](#) and [ELR_EL3](#) on exception return is synchronized.
- If the PE enters Debug state before the first instruction after an Exception return from EL3 to Non-secure state, any pending Halting debug event completes execution.
- The GIC behavior that allocates interrupts to FIQ or IRQ changes simultaneously with leaving the EL3 Exception level.
- Behavior of accessing the banked copies of the stack pointer using the SP register name for loads, stores and data processing instructions.
- Exit from Debug state.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bits [10:7]

Reserved, RES0.

nAA, bit [6]

When FEAT_LSE2 is implemented:

Non-aligned access. This bit controls generation of Alignment faults at EL3 under certain conditions.

| nAA | Meaning |
|-----|--|
| 0b0 | LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, and STLURH generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes for accesses. |
| 0b1 | This control bit does not cause LDAPR, LDAPRH, LDAPUR, LDAPURH, LDAPURSH, LDAPURSW, LDAR, LDARH, LDLAR, LDLARH, STLLR, STLLRH, STLR, STLRH, STLUR, or STLURH to generate an Alignment fault if all bytes being accessed are not within a single 16-byte quantity, aligned to 16 bytes. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [5:4]

Reserved, RES1.

SA, bit [3]

SP Alignment check enable. When set to 1, if a load or store instruction executed at EL3 uses the SP as the base address and the SP is not aligned to a 16-byte boundary, then a SP alignment fault exception is generated. For more information, see 'SP alignment checking'.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

C, bit [2]

Cacheability control, for data accesses.

| C | Meaning |
|----------|--|
| 0b0 | All data access to Normal memory from EL3, and all Normal memory accesses to the EL3 translation tables, are Non-cacheable for all levels of data and unified cache. |
| 0b1 | This control has no effect on the Cacheability of: <ul style="list-style-type: none"> Data access to Normal memory from EL3. Normal memory accesses to the EL3 translation tables. |

This bit has no effect on the EL1&0, EL2, or EL2&0 translation regimes.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL3.

| A | Meaning |
|----------|---|
| 0b0 | Alignment fault checking disabled when executing at EL3. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed. |
| 0b1 | Alignment fault checking enabled when executing at EL3. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception. |

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL3 stage 1 address translation. Possible values of this bit are:

| M | Meaning |
|----------|--|
| 0b0 | EL3 stage 1 address translation disabled. See the SCTLR_EL3.I field for the behavior of instruction accesses to Normal memory. |
| 0b1 | EL3 stage 1 address translation enabled. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Accessing SCTLR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SCTLR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SCTLR_EL3;

```

MSR SCTLR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCTLR_EL3 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCXTNUM_EL0, EL0 Read/Write Software Context Number

The SCXTNUM_EL0 characteristics are:

Purpose

Provides a number that can be used to separate out different context numbers with the EL0 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

Configuration

This register is present only when FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented. Otherwise, direct accesses to SCXTNUM_EL0 are UNDEFINED.

Attributes

SCXTNUM_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Software Context Number | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Software Context Number | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Software Context Number. A number to identify the context within the EL0 exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCXTNUM_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SCXTNUM_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.TSCXT == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.EnSCXT == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGTR_EL2.SCXTNUM_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.TSCXT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return SCXTNUM_EL0;
        elsif PSTATE.EL == EL1 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
                UNDEFINED;
            elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
                AArch64.SystemAccessTrap(EL2, 0x18);
            elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.SCXTNUM_EL0 == '1'
then
                AArch64.SystemAccessTrap(EL2, 0x18);
            elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
                else
                    return SCXTNUM_EL0;
            elsif PSTATE.EL == EL2 then
                if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
                    UNDEFINED;
                elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.SystemAccessTrap(EL3, 0x18);
                    else
                        return SCXTNUM_EL0;
            elsif PSTATE.EL == EL3 then
                return SCXTNUM_EL0;

```

MSR SCXTNUM_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTL_EL1.TSCXT == '1' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.EnSCXT == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGWTR_EL2.SCXTNUM_EL0 == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTL_EL2.TSCXT == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                SCXTNUM_EL0 = X[t];
        elsif PSTATE.EL == EL1 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
                UNDEFINED;
            elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
                AArch64.SystemAccessTrap(EL2, 0x18);
            elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.SCXTNUM_EL0 == '1'
then
                AArch64.SystemAccessTrap(EL2, 0x18);
            elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
                else
                    SCXTNUM_EL0 = X[t];
        elsif PSTATE.EL == EL2 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
                UNDEFINED;
            elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.SystemAccessTrap(EL3, 0x18);
            else
                SCXTNUM_EL0 = X[t];
        elsif PSTATE.EL == EL3 then
            SCXTNUM_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCXTNUM_EL1, EL1 Read/Write Software Context Number

The SCXTNUM_EL1 characteristics are:

Purpose

Provides a number that can be used to separate out different context numbers with the EL1 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

Configuration

This register is present only when FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented. Otherwise, direct accesses to SCXTNUM_EL1 are UNDEFINED.

Attributes

SCXTNUM_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Software Context Number | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Software Context Number | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Software Context Number. A number to identify the context within the EL1 exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCXTNUM_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SCXTNUM_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.SCXTNUM_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            return NVMem[0x188];
        else
            return SCXTNUM_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            return SCXTNUM_EL2;
        else
            return SCXTNUM_EL1;
    elsif PSTATE.EL == EL3 then
        return SCXTNUM_EL1;

```

MSR SCXTNUM_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.SCXTNUM_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            NVMem[0x188] = X[t];
        else
            SCXTNUM_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            SCXTNUM_EL2 = X[t];
        else
            SCXTNUM_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        SCXTNUM_EL1 = X[t];

```

MRS <Xt>, SCXTNUM_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1101 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x188];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return SCXTNUM_EL1;
        else
            UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return SCXTNUM_EL1;
    else
        UNDEFINED;

```

MSR SCXTNUM_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1101 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x188] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                SCXTNUM_EL1 = X[t];
        else
            UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        SCXTNUM_EL1 = X[t];
    else
        UNDEFINED;

```


SCXTNUM_EL2, EL2 Read/Write Software Context Number

The SCXTNUM_EL2 characteristics are:

Purpose

Provides a number that can be used to separate out different context numbers with the EL2 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

Configuration

This register is present only when FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented. Otherwise, direct accesses to SCXTNUM_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

SCXTNUM_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Software Context Number | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Software Context Number | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Software Context Number. A number to identify the context within the EL2 exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCXTNUM_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic SCXTNUM_EL2 or SCXTNUM_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SCXTNUM_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1101 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return SCXTNUM_EL2;
elsif PSTATE.EL == EL3 then
    return SCXTNUM_EL2;

```

MSR SCXTNUM_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1101 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SCXTNUM_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL2 = X[t];

```

MRS <Xt>, SCXTNUM_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.SCXTNUM_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            return NVMem[0x188];
        else
            return SCXTNUM_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        elsif HCR_EL2.E2H == '1' then
            return SCXTNUM_EL2;
        else
            return SCXTNUM_EL1;
    elsif PSTATE.EL == EL3 then
        return SCXTNUM_EL1;

```

MSR SCXTNUM_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.EnSCXT == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.SCXTNUM_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            NVMem[0x188] = X[t];
        else
            SCXTNUM_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnSCXT == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.EnSCXT == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            elsif HCR_EL2.E2H == '1' then
                SCXTNUM_EL2 = X[t];
            else
                SCXTNUM_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        SCXTNUM_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCXTNUM_EL3, EL3 Read/Write Software Context Number

The SCXTNUM_EL3 characteristics are:

Purpose

Provides a number that can be used to separate out different context numbers with the EL3 exception level, for the purpose of protecting against side-channels using branch prediction and similar resources.

Configuration

This register is present only when EL3 is implemented and (FEAT_CSV2_2 is implemented or FEAT_CSV2_1p2 is implemented). Otherwise, direct accesses to SCXTNUM_EL3 are UNDEFINED.

Attributes

SCXTNUM_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Software Context Number | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Software Context Number | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Software Context Number. A number to identify the context within the EL3 exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SCXTNUM_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SCXTNUM_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1101 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SCXTNUM_EL3;

```

MSR SCXTNUM_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1101 | 0b0000 | 0b111 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCXTNUM_EL3 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SDER32_EL2, AArch32 Secure Debug Enable Register

The SDER32_EL2 characteristics are:

Purpose

Allows access to the AArch32 register [SDER](#) from Secure EL2 and EL3 only.

Configuration

AArch64 System register SDER32_EL2 bits [63:0] are architecturally mapped to AArch64 System register [SDER32_EL3\[63:0\]](#) when EL3 is implemented.

AArch64 System register SDER32_EL2 bits [31:0] are architecturally mapped to AArch32 System register [SDER\[31:0\]](#).

This register is present only when EL2 is implemented, FEAT_SEL2 is implemented and EL1 is capable of using AArch32. Otherwise, direct accesses to SDER32_EL2 are UNDEFINED.

This register is ignored by the PE when one or more of the following are true:

- The PE is in Non-secure state.
- EL1 is using AArch64.

Attributes

SDER32_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|--------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | SUNIDEN | | SUIDEN |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:2]

Reserved, RES0.

SUNIDEN, bit [1]

Secure User Non-Invasive Debug Enable.

| SUNIDEN | Meaning |
|---------|--|
| 0b0 | This bit does not affect Performance Monitors event counting at Secure EL0. |
| 0b1 | If EL1 is using AArch32, Performance Monitors event counting is allowed in Secure EL0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SUIDEN, bit [0]

Secure User Invasive Debug Enable.

| SUIDEN | Meaning |
|--------|--|
| 0b0 | This bit does not affect the generation of debug exceptions at Secure EL0. |
| 0b1 | If EL1 is using AArch32, debug exceptions from Secure EL0 are enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SDER32_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SDER32_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return SDER32_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        return SDER32_EL2;

```

MSR SDER32_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0011 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SDER32_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        SDER32_EL2 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SDER32_EL3, AArch32 Secure Debug Enable Register

The SDER32_EL3 characteristics are:

Purpose

Allows access to the AArch32 register [SDER](#) from AArch64 state only. Its value has no effect on execution in AArch64 state.

Configuration

AArch64 System register `SDER32_EL3` bits [63:0] are architecturally mapped to AArch64 System register `SDER32_EL2` [63:0] when EL2 is implemented and FEAT_SEL2 is implemented.

AArch64 System register SDER32_EL3 bits [31:0] are architecturally mapped to AArch32 System register [SDER\[31:0\]](#).

This register is present only when EL3 is implemented and EL1 is capable of using AArch32. Otherwise, direct accesses to `SDER32_EL3` are UNDEFINED.

This register is ignored by the PE when one or more of the following are true:

- The PE is in Non-secure state.
- EL1 is using AArch64.

Attributes

SDER32_EL3 is a 64-bit register.

Field descriptions

63 62 61 60 59 58 57 56 55 54 53 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32

RES0

RES0 SUNIDEN SUIDEN

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Bits [63:2]

Reserved, RES0.

SUNIDEN, bit [1]

Secure User Non-Invasive Debug Enable.

| SUNIDEN | Meaning |
|---------|--|
| 0b0 | This bit does not affect Performance Monitors event counting at Secure EL0. |
| 0b1 | If EL1 is using AArch32, Performance Monitors event counting is allowed in Secure EL0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SUIDEN, bit [0]

Secure User Invasive Debug Enable.

| SUIDEN | Meaning |
|--------|--|
| 0b0 | This bit does not affect the generation of debug exceptions at Secure EL0. |
| 0b1 | If EL1 is using AArch32, debug exceptions from Secure EL0 are enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SDER32_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SDER32_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0001 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SDER32_EL3;
```

MSR SDER32_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0001 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SDER32_EL3 = X[t];
```

SMCR_EL1, SME Control Register (EL1)

The SMCR_EL1 characteristics are:

Purpose

This register controls aspects of Streaming SVE that are visible at Exception levels EL1 and EL0.

Configuration

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMCR_EL1 are UNDEFINED.

This register has no effect if the PE is not in Streaming SVE mode.

When [HCR_EL2](#).{E2H, TGE} == {1, 1} and EL2 is enabled in the current Security state, this register has no effect on execution at EL0 and EL1.

Attributes

SMCR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| FA64 | | | | | | | | | | | | | | | | RAZ/WI | | | | | | | | LEN | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

FA64, bit [31]

When FEAT_SME_FA64 is implemented:

Controls whether execution of an A64 instruction is considered legal when the PE is in Streaming SVE mode.

| FA64 | Meaning |
|------|---|
| 0b0 | This control does not cause any instruction to be treated as legal in Streaming SVE mode. |
| 0b1 | This control causes all implemented A64 instructions to be treated as legal in Streaming SVE mode at EL1 and EL0, if they are treated as legal at more privileged Exception levels in the current Security state. |

Arm recommends that portable SME software should not rely on this optional feature, and that operating systems should provide a means to test for compliance with this recommendation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [30:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Effective Streaming SVE Vector Length (SVL).

Constrains the effective Streaming SVE vector register length for EL1 and EL0 to (LEN+1)*128 bits. SVL only takes effect when the PE is in Streaming SVE mode.

An implementation is permitted to include any set of Streaming SVE vector lengths that are powers of two, from 128 bits to 2048 bits inclusive.

For all purposes other than returning the result of a direct read of SMCR_EL1, this field selects the effective vector length as follows:

- If the requested length is smaller than the minimum implemented Streaming SVE vector length, then the minimum implemented Streaming SVE vector length is used.
- If the requested length is larger than the effective vector length at the next more privileged Exception level in the current Security state, if any, then the effective vector length at the more privileged Exception level is used.
- If the requested length is not implemented, then the requested length rounded down to the nearest implemented Streaming SVE vector length is used.
- Otherwise, the requested length is used.

An indirect read of SMCR_EL1.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMCR_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic SMCR_EL1 or SMCR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SMCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
        end if
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x1F0];
    else
        return SMCR_EL1;
    end if
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
        end if
    elsif HCR_EL2.E2H == '1' then
        return SMCR_EL2;
    else
        return SMCR_EL1;
    end if
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        return SMCR_EL1;
    end if
end if

```

MSR SMCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
        endif
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x1F0] = X[t];
    else
        SMCR_EL1 = X[t];
endif
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
        endif
    elsif HCR_EL2.E2H == '1' then
        SMCR_EL2 = X[t];
    else
        SMCR_EL1 = X[t];
endif
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL1 = X[t];
    endif
endif

```

MRS <Xt>, SMCR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0001 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x1F0];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
            UNDEFINED;
        elsif CPTR_EL3.SMEN == 'x0' then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x1D);
            else
                return SMCR_EL1;
        else
            UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        if CPTR_EL3.ESM == '0' then
            AArch64.SystemAccessTrap(EL3, 0x1D);
        else
            return SMCR_EL1;
    else
        UNDEFINED;

```

MSR SMCR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0001 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x1F0] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
            UNDEFINED;
        elsif CPTR_EL2.SMEN == 'x0' then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x1D);
        else
            SMCR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        if CPTR_EL3.ESM == '0' then
            AArch64.SystemAccessTrap(EL3, 0x1D);
        else
            SMCR_EL1 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SMCR_EL2, SME Control Register (EL2)

The SMCR_EL2 characteristics are:

Purpose

This register controls aspects of Streaming SVE that are visible at Exception levels EL2, EL1, and EL0.

Configuration

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMCR_EL2 are UNDEFINED.

This register has no effect if the PE is not in Streaming SVE mode, or if EL2 is not enabled in the current Security state.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

SMCR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| FA64 | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | RAZ/WI | | | | | | | | LEN | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

FA64, bit [31] When FEAT_SME_FA64 is implemented:

Controls whether execution of an A64 instruction is considered legal when the PE is in Streaming SVE mode.

| FA64 | Meaning |
|------|--|
| 0b0 | This control does not cause any instruction to be treated as legal in Streaming SVE mode. |
| 0b1 | This control causes all implemented A64 instructions to be treated as legal in Streaming SVE mode at EL2, if they are treated as legal at EL3. |

Arm recommends that portable SME software should not rely on this optional feature, and that operating systems should provide a means to test for compliance with this recommendation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [30:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Effective Streaming SVE Vector Length (SVL).

Constrains the effective Streaming SVE vector register length for EL2, EL1, and EL0 to (LEN+1)*128 bits when EL2 is enabled in the current Security state. SVL only takes effect when the PE is in Streaming SVE mode.

An implementation is permitted to include any set of Streaming SVE vector lengths that are powers of two, from 128 bits to 2048 bits inclusive.

For all purposes other than returning the result of a direct read of SMCR_EL2, this field selects the effective vector length as follows:

- If the requested length is smaller than the minimum implemented Streaming SVE vector length, then the minimum implemented Streaming SVE vector length is used.
- If the requested length is larger than the effective vector length at the next more privileged Exception level in the current Security state, if any, then the effective vector length at the more privileged Exception level is used.
- If the requested length is not implemented, then the requested length rounded down to the nearest implemented Streaming SVE vector length is used.
- Otherwise, the requested length is used.

An indirect read of SMCR_EL2.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMCR_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic SMCR_EL2 or SMCR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SMCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        return SMCR_EL2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        return SMCR_EL2;

```

MSR SMCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL2 = X[t];

```

MRS <Xt>, SMCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b110 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            return NVMem[0x1F0];
        else
            return SMCR_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
            UNDEFINED;
        elsif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        elsif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x1D);
        elsif HCR_EL2.E2H == '1' then
            return SMCR_EL2;
        else
            return SMCR_EL1;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.ESM == '0' then
            AArch64.SystemAccessTrap(EL3, 0x1D);
        else
            return SMCR_EL1;

```

MSR SMCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
        elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
            NVMem[0x1F0] = X[t];
        else
            SMCR_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
            UNDEFINED;
        elsif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        elsif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x1D);
        elsif HCR_EL2.E2H == '1' then
            SMCR_EL2 = X[t];
        else
            SMCR_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.ESM == '0' then
            AArch64.SystemAccessTrap(EL3, 0x1D);
        else
            SMCR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SMCR_EL3, SME Control Register (EL3)

The SMCR_EL3 characteristics are:

Purpose

This register controls aspects of Streaming SVE that are visible at all Exception levels.

Configuration

This register is present only when FEAT_SME is implemented and EL3 is implemented. Otherwise, direct accesses to SMCR_EL3 are UNDEFINED.

This register has no effect if the PE is not in Streaming SVE mode.

Attributes

SMCR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | FA64 | | | | | | | | RES0 | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | FA64 | | | | | | | | RES0 | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

FA64, bit [31]

When FEAT_SME_FA64 is implemented:

Controls whether execution of an A64 instruction is considered legal when the PE is in Streaming SVE mode.

| FA64 | Meaning |
|------|---|
| 0b0 | This control does not cause any instruction to be treated as legal in Streaming SVE mode. |
| 0b1 | This control causes all implemented A64 instructions to be treated as legal in Streaming SVE mode at EL3. |

Arm recommends that portable SME software should not rely on this optional feature, and that operating systems should provide a means to test for compliance with this recommendation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [30:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Effective Streaming SVE Vector Length (SVL).

Constrains the effective Streaming SVE vector register length for all Exception levels to (LEN+1)*128 bits. SVL only takes effect when the PE is in Streaming SVE mode.

An implementation is permitted to include any set of Streaming SVE vector lengths that are powers of two, from 128 bits to 2048 bits inclusive.

For all purposes other than returning the result of a direct read of SMCR_EL3, this field selects the effective vector length as follows:

- If the requested length is smaller than the minimum implemented Streaming SVE vector length, then the minimum implemented Streaming SVE vector length is used.
- If the requested length is not implemented, then the requested length rounded down to the nearest implemented Streaming SVE vector length is used.
- Otherwise, the requested length is used.

An indirect read of SMCR_EL3.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SMCR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        return SMCR_EL3;
    end
end
    
```

MSR SMCR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0010 | 0b110 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        SMCR_EL3 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SMIDR_EL1, Streaming Mode Identification Register

The SMIDR_EL1 characteristics are:

Purpose

Provides additional identification mechanisms for scheduling purposes, for a PE that supports Streaming SVE mode.

Configuration

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMIDR_EL1 are UNDEFINED.

Attributes

SMIDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|------|------|----|----|----|----------|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | |
| Implementer | | | | | | | | Revision | | | | | | | | SMPS | RES0 | | | | Affinity | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:32]

Reserved, RES0.

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

| Implementer | Meaning |
|-------------|--|
| 0x00 | Reserved for software use. |
| 0x41 | Arm Limited. |
| 0x42 | Broadcom Corporation. |
| 0x43 | Cavium Inc. |
| 0x44 | Digital Equipment Corporation. |
| 0x46 | Fujitsu Ltd. |
| 0x49 | Infineon Technologies AG. |
| 0x4D | Motorola or Freescale Semiconductor Inc. |
| 0x4E | NVIDIA Corporation. |
| 0x50 | Applied Micro Circuits Corporation. |
| 0x51 | Qualcomm Inc. |
| 0x56 | Marvell International Ltd. |
| 0x69 | Intel Corporation. |
| 0xC0 | Ampere Computing. |

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

It is not required that this value is the same as the value of [MIDR_EL1.Implementer](#).

Access to this field is **RO**.

Revision, bits [23:16]

Revision number for the Streaming Mode Compute Unit (SMCU).

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SMPS, bit [15]

Indicates support for Streaming SVE mode execution priority.

| SMPS | Meaning |
|------|---------------------------------|
| 0b0 | Priority control not supported. |
| 0b1 | Priority control supported. |

Bits [14:12]

Reserved, RES0.

Affinity, bits [11:0]

The SMCU affinity of the accessing PE.

- A value of zero indicates that the PE's implementation of Streaming SVE mode is not shared with other PEs.
- Otherwise, the value identifies which SMCU is associated with this PE. The Affinity value associated with each SMCU is unique within the system as a whole.

Accessing SMIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SMIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b001 | 0b0000 | 0b0000 | 0b110 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TID1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return SMIDR_EL1;
elseif PSTATE.EL == EL2 then
    return SMIDR_EL1;
elseif PSTATE.EL == EL3 then
    return SMIDR_EL1;

```

SMPRI_EL1, Streaming Mode Priority Register

The SMPRI_EL1 characteristics are:

Purpose

Configures the streaming execution priority for instructions executed on a shared Streaming Mode Compute Unit (SMCU) when the PE is in Streaming SVE mode at any Exception Level.

Configuration

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMPRI_EL1 are UNDEFINED.

When [SMIDR_EL1](#).SMPS is '0', this register is RES0.

Attributes

SMPRI_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:4]

Reserved, RES0.

Priority, bits [3:0]

Streaming execution priority value.

Either this value is used directly, or it is mapped into an effective priority value using [SMPRMAP_EL2](#).

This value is used directly when any of the following are true:

- The current Exception level is EL3 or EL2.
- The current Exception level is EL1 or EL0, if EL2 is implemented and enabled in the current Security state and [HCRX_EL2](#).SMPME is '0'.
- The current Exception level is EL1 or EL0, if EL2 is either not implemented or not enabled in the current Security state.

The precise meaning and behavior of each streaming execution priority value is IMPLEMENTATION DEFINED.

In an implementation that shares execution resources between PEs, higher priority values are allocated more processing resource than other PEs configured with lower priority values in the same Priority domain.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMPRI_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SMPRI_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.nSMPRI_EL1 == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return SMPRI_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return SMPRI_EL1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return SMPRI_EL1;

```

MSR SMPRI_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.nSMPRI_EL1 == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SMPRI_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SMPRI_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.ESM == '0' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            SMPRI_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SMPRIMAP_EL2, Streaming Mode Priority Mapping Register

The SMPRIMAP_EL2 characteristics are:

Purpose

Maps the value in [SMPRI_EL1](#) to a streaming execution priority value for instructions executed at EL1 and EL0 in the same Security states as EL2.

Configuration

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SMPRIMAP_EL2 are UNDEFINED.

When [SMIDR_EL1](#).SMPS is '0', this register is RES0.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

SMPRIMAP_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| P15 | | | | P14 | | | | P13 | | | | P12 | | | | P11 | | | | P10 | | | P9 | | | | P8 | | | | |
| P7 | | | | P6 | | | | P5 | | | | P4 | | | | P3 | | | | P2 | | | P1 | | | | P0 | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

When all of the following are true, the value in [SMPRI_EL1](#) is mapped to a streaming execution priority using this register:

- The current Exception level is EL1 or EL0.
- EL2 is implemented and enabled in the current Security state.
- [HCRX_EL2](#).SMPME is '1'.

Otherwise, [SMPRI_EL1](#) holds the streaming execution priority value.

P15, bits [63:60]

Priority Mapping Entry 15. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#).Priority value is '15'.

This value is the highest streaming execution priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P14, bits [59:56]

Priority Mapping Entry 14. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#).Priority value is '14'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P13, bits [55:52]

Priority Mapping Entry 13. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '13'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P12, bits [51:48]

Priority Mapping Entry 12. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '12'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P11, bits [47:44]

Priority Mapping Entry 11. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '11'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P10, bits [43:40]

Priority Mapping Entry 10. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '10'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P9, bits [39:36]

Priority Mapping Entry 9. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '9'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P8, bits [35:32]

Priority Mapping Entry 8. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '8'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P7, bits [31:28]

Priority Mapping Entry 7. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '7'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P6, bits [27:24]

Priority Mapping Entry 6. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '6'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P5, bits [23:20]

Priority Mapping Entry 5. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '5'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P4, bits [19:16]

Priority Mapping Entry 4. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '4'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P3, bits [15:12]

Priority Mapping Entry 3. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '3'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P2, bits [11:8]

Priority Mapping Entry 2. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '2'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P1, bits [7:4]

Priority Mapping Entry 1. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '1'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P0, bits [3:0]

Priority Mapping Entry 0. This entry is used when priority mapping is supported and enabled, and the [SMPRI_EL1](#). Priority value is '0'.

This value is the lowest streaming execution priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SMPRIMAP_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SMPRIMAP_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x1E8];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return SMPRIMAP_EL2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return SMPRIMAP_EL2;

```

MSR SMPRIMAP_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x1E8] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SMPRIMAP_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        SMPRIMAP_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SP_EL0, Stack Pointer (EL0)

The SP_EL0 characteristics are:

Purpose

Holds the stack pointer associated with EL0. At higher Exception levels, this is used as the current stack pointer when the value of [SPSel.SP](#) is 0.

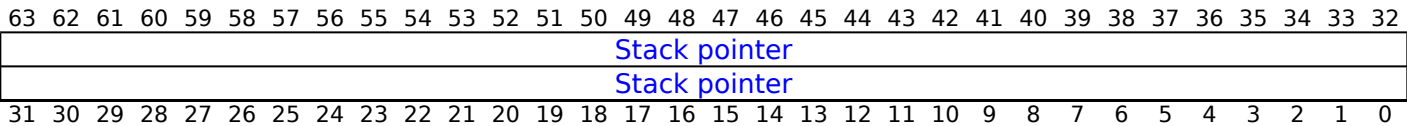
Configuration

There are no configuration notes.

Attributes

SP_EL0 is a 64-bit register.

Field descriptions



Bits [63:0]

Stack pointer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SP_EL0

When the value of PSTATE.SP is 0, this register is accessible at all Exception levels as the current stack pointer.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SP_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if PSTATE.SP == '0' then
        UNDEFINED;
    else
        return SP_EL0;
elsif PSTATE.EL == EL2 then
    if PSTATE.SP == '0' then
        UNDEFINED;
    else
        return SP_EL0;
elsif PSTATE.EL == EL3 then
    if PSTATE.SP == '0' then
        UNDEFINED;
    else
        return SP_EL0;

```

MSR SP_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if PSTATE.SP == '0' then
        UNDEFINED;
    else
        SP_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if PSTATE.SP == '0' then
        UNDEFINED;
    else
        SP_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.SP == '0' then
        UNDEFINED;
    else
        SP_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SP_EL1, Stack Pointer (EL1)

The SP_EL1 characteristics are:

Purpose

Holds the stack pointer associated with EL1. When executing at EL1, the value of [SPSel.SP](#) determines the current stack pointer:

| SPSel.SP | Current stack pointer |
|----------|------------------------|
| 0b0 | SP_EL0 |
| 0b1 | SP_EL1 |

Configuration

There are no configuration notes.

Attributes

SP_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Stack pointer | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Stack pointer | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:0]

Stack pointer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SP_EL1

This accessibility information only applies to accesses using the MRS or MSR instructions.

When the value of [SPSel.SP](#) is 1, this register is also accessible at EL1 as the current stack pointer.

Note

When the value of [SPSel.SP](#) is 0, [SP_EL0](#) is used as the current stack pointer at all Exception levels.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SP_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0100 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x240];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return SP_EL1;
elsif PSTATE.EL == EL3 then
    return SP_EL1;

```

MSR SP_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0100 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x240] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    SP_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    SP_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SP_EL2, Stack Pointer (EL2)

The SP_EL2 characteristics are:

Purpose

Holds the stack pointer associated with EL2. When executing at EL2, the value of [SPSel.SP](#) determines the current stack pointer:

| SPSel.SP | Current stack pointer |
|----------|------------------------|
| 0b0 | SP_EL0 |
| 0b1 | SP_EL2 |

Configuration

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

SP_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Stack pointer | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Stack pointer | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:0]

Stack pointer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SP_EL2

This accessibility information only applies to accesses using the MRS or MSR instructions.

When the value of [SPSel.SP](#) is 1, this register is also accessible at EL2 as the current stack pointer.

Note

When the value of [SPSel.SP](#) is 0, [SP_EL0](#) is used as the current stack pointer at all Exception levels.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SP_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0100 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SP_EL2;

```

MSR SP_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0100 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SP_EL2 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SP_EL3, Stack Pointer (EL3)

The SP_EL3 characteristics are:

Purpose

Holds the stack pointer associated with EL3. When executing at EL3, the value of [SPSel.SP](#) determines the current stack pointer:

| SPSel.SP | Current stack pointer |
|----------|------------------------|
| 0b0 | SP_EL0 |
| 0b1 | SP_EL3 |

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to SP_EL3 are UNDEFINED.

Attributes

SP_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Stack pointer | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Stack pointer | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:0]

Stack pointer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SP_EL3

This register is not accessible using MRS and MSR instructions.

When the value of [SPSel.SP](#) is 1, this register is accessible at EL3 as the current stack pointer.

Note

When the value of [SPSel.SP](#) is 0, [SP_EL0](#) is used as the current stack pointer at all Exception levels.

SPSel, Stack Pointer Select

The SPSel characteristics are:

Purpose

Allows the Stack Pointer to be selected between SP_EL0 and SP_ELx.

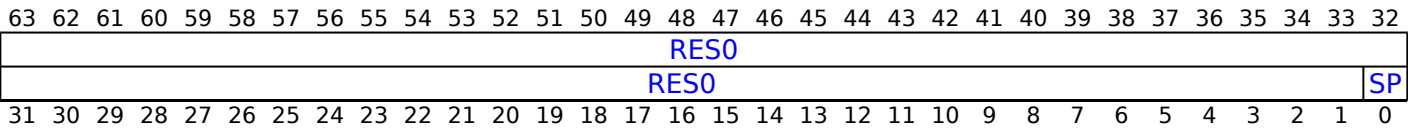
Configuration

There are no configuration notes.

Attributes

SPSel is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

SP, bit [0]

Stack pointer to use. Possible values of this bit are:

| SP | Meaning |
|-----|-------------------------------------|
| 0b0 | Use SP_EL0 at all Exception levels. |
| 0b1 | Use SP_ELx for Exception level ELx. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

Accessing SPSel

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SPSel

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    return Zeros(63):PSTATE.SP;
elsif PSTATE.EL == EL2 then
    return Zeros(63):PSTATE.SP;
elsif PSTATE.EL == EL3 then
    return Zeros(63):PSTATE.SP;

```

MSR SPSel, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    PSTATE.SP = X[t]<0>;
elsif PSTATE.EL == EL2 then
    PSTATE.SP = X[t]<0>;
elsif PSTATE.EL == EL3 then
    PSTATE.SP = X[t]<0>;

```

MSR SPSel, #<imm>

| op0 | op1 | CRn | op2 |
|------|-------|--------|-------|
| 0b00 | 0b000 | 0b0100 | 0b101 |

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_abt, Saved Program Status Register (Abort mode)

The SPSR_abt characteristics are:

Purpose

Holds the saved process state when an exception is taken to Abort mode.

Configuration

AArch64 System register SPSR_abt bits [31:0] are architecturally mapped to AArch32 System register [SPSR_abt\[31:0\]](#).

If EL1 only supports execution in AArch64 state, this register is RES0 from EL2 and EL3.

Attributes

SPSR_abt is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|---------|----|------|-----|-----|----|----|---------|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | Z | C | V | Q | IT[1:0] | J | SSBS | PAN | DIT | IL | GE | IT[7:2] | E | A | I | F | T | M[4:0] | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Abort mode, and copied to PSTATE.N on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Abort mode, and copied to PSTATE.Z on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Abort mode, and copied to PSTATE.C on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Abort mode, and copied to PSTATE.V on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Abort mode, and copied to PSTATE.Q on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Abort mode, and copied to PSTATE.IT on executing an exception return operation in Abort mode.

SPSR_abt.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_abt[26:25].
- IT[7:2] is SPSR_abt[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Abort mode, and copied to PSTATE.SSBS on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Abort mode, and copied to PSTATE.PAN on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Abort mode, and copied to PSTATE.DIT on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Abort mode, and copied to PSTATE.IL on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Abort mode, and copied to PSTATE.GE on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Abort mode, and copied to PSTATE.E on executing an exception return operation in Abort mode.

If the implementation does not support big-endian operation, SPSR_abt.E is RES0. If the implementation does not support little-endian operation, SPSR_abt.E is RES1. On executing an exception return operation in Abort mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_abt.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_abt.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Abort mode, and copied to PSTATE.A on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Abort mode, and copied to PSTATE.I on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Abort mode, and copied to PSTATE.F on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Abort mode, and copied to PSTATE.T on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Abort mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Abort mode.

| M[4:0] | Meaning |
|---------|-------------|
| 0b10000 | User. |
| 0b10001 | FIQ. |
| 0b10010 | IRQ. |
| 0b10011 | Supervisor. |
| 0b10111 | Abort. |
| 0b11011 | Undefined. |
| 0b11111 | System. |

Other values are reserved. If SPSR_abt.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Abort mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_abt

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SPSR_abt

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0100 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    return SPSR_abt;
elseif PSTATE.EL == EL3 then
    return SPSR_abt;

```

MSR SPSR_abt, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0100 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    SPSR_abt = X[t];
elseif PSTATE.EL == EL3 then
    SPSR_abt = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_EL1, Saved Program Status Register (EL1)

The SPSR_EL1 characteristics are:

Purpose

Holds the saved process state when an exception is taken to EL1.

Configuration

AArch64 System register SPSR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [SPSR_svc\[31:0\]](#).

Attributes

SPSR_EL1 is a 64-bit register.

Field descriptions

When AArch32 is supported and exception taken from AArch32 state:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|---------|-----|------|-----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|------|----|----|----|--------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | Z | C | V | Q | IT[1:0] | DIT | SSBS | PAN | SS | IL | GE | | | | IT[7:2] | | | | E | A | I | F | T | M[4] | | | | M[3:0] | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

An exception return from EL1 using AArch64 makes SPSR_EL1 become UNKNOWN.

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL1, and copied to PSTATE.N on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL1, and copied to PSTATE.Z on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL1, and copied to PSTATE.C on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL1, and copied to PSTATE.V on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to EL1, and copied to PSTATE.Q on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to EL1, and copied to PSTATE.IT on executing an exception return operation in EL1.

SPSR_EL1.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_EL1[26:25].
- IT[7:2] is SPSR_EL1[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL1, and copied to PSTATE.DIT on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL1, and copied to PSTATE.SSBS on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]**When FEAT_PAN is implemented:**

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL1, and copied to PSTATE.PAN on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL1, and conditionally copied to PSTATE.SS on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL1, and copied to PSTATE.IL on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to EL1, and copied to PSTATE.GE on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to EL1, and copied to PSTATE.E on executing an exception return operation in EL1.

If the implementation does not support big-endian operation, SPSR_EL1.E is RES0. If the implementation does not support little-endian operation, SPSR_EL1.E is RES1. On executing an exception return operation in EL1, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_EL1.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_EL1.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL1, and copied to PSTATE.A on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL1, and copied to PSTATE.I on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL1, and copied to PSTATE.F on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to EL1, and copied to PSTATE.T on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4], bit [4]

Execution state. Set to 0b1, the value of PSTATE.nRW, on taking an exception to EL1 from AArch32 state, and copied to PSTATE.nRW on executing an exception return operation in EL1.

| M[4] | Meaning |
|-------------|--------------------------|
| 0b1 | AArch32 execution state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch32 Mode. Set to the value of PSTATE.M[3:0] on taking an exception to EL1, and copied to PSTATE.M[3:0] on executing an exception return operation in EL1.

| M[3:0] | Meaning |
|---------------|----------------|
| 0b0000 | User. |
| 0b0001 | FIQ. |
| 0b0010 | IRQ. |
| 0b0011 | Supervisor. |
| 0b0111 | Abort. |
| 0b1011 | Undefined. |
| 0b1111 | System. |

Other values are reserved. If SPSR_EL1.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL1 is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When exception taken from AArch64 state:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|------|-----|-----|-----|-----|----|----|------|----|----|----|--------|------|-------|----|----|----|----|------|------|--------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | Z | C | V | RES0 | TCO | DIT | UAO | PAN | SS | IL | RES0 | | | | ALLINT | SSBS | BTYPE | D | A | I | F | RES0 | M[4] | M[3:0] | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

An exception return from EL1 using AArch64 makes SPSR_EL1 become UNKNOWN.

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL1, and copied to PSTATE.N on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL1, and copied to PSTATE.Z on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL1, and copied to PSTATE.C on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL1, and copied to PSTATE.V on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

TCO, bit [25]

When FEAT_MTE is implemented:

Tag Check Override. Set to the value of PSTATE.TCO on taking an exception to EL1, and copied to PSTATE.TCO on executing an exception return operation in EL1.

When FEAT_MTE2 is not implemented, it is CONSTRAINED UNPREDICTABLE whether this field is RES0 or behaves as if FEAT_MTE is implemented.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL1, and copied to PSTATE.DIT on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UAO, bit [23]

When FEAT_UAO is implemented:

User Access Override. Set to the value of PSTATE.UAO on taking an exception to EL1, and copied to PSTATE.UAO on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL1, and copied to PSTATE.PAN on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL1, and conditionally copied to PSTATE.SS on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL1, and copied to PSTATE.IL on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:14]

Reserved, RES0.

ALLINT, bit [13]

When FEAT_NMI is implemented:

All IRQ or FIQ interrupts mask. Set to the value of PSTATE.ALLINT on taking an exception to EL1, and copied to PSTATE.ALLINT on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [12]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL1, and copied to PSTATE.SSBS on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BTYPE, bits [11:10]

When FEAT_BTI is implemented:

Branch Type Indicator. Set to the value of PSTATE.BTYPE on taking an exception to EL1, and copied to PSTATE.BTYPE on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D, bit [9]

Debug exception mask. Set to the value of PSTATE.D on taking an exception to EL1, and copied to PSTATE.D on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL1, and copied to PSTATE.A on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL1, and copied to PSTATE.I on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL1, and copied to PSTATE.F on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

M[4], bit [4]

Execution state. Set to 0b0, the value of PSTATE.nRW, on taking an exception to EL1 from AArch64 state, and copied to PSTATE.nRW on executing an exception return operation in EL1.

| M[4] | Meaning |
|-------------|--------------------------|
| 0b0 | AArch64 execution state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch64 Exception level and selected Stack Pointer.

| M[3:0] | Meaning |
|---------------|----------------|
| 0b0000 | EL0t. |
| 0b0100 | EL1t. |
| 0b0101 | EL1h. |

Other values are reserved. If SPSR_EL1.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL1 is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The bits in this field are interpreted as follows:

- M[3:2] is set to the value of PSTATE.EL on taking an exception to EL1 and copied to PSTATE.EL on executing an exception return operation in EL1.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on taking an exception to EL1 and copied to PSTATE.SP on executing an exception return operation in EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic SPSR_EL1 or SPSR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SPSR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x160];
    else
        return SPSR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return SPSR_EL2;
    else
        return SPSR_EL1;
elsif PSTATE.EL == EL3 then
    return SPSR_EL1;

```

MSR SPSR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x160] = X[t];
    else
        SPSR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        SPSR_EL2 = X[t];
    else
        SPSR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    SPSR_EL1 = X[t];

```

MRS <Xt>, SPSR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x160];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return SPSR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return SPSR_EL1;
    else
        UNDEFINED;

```

MSR SPSR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x160] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        SPSR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        SPSR_EL1 = X[t];
    else
        UNDEFINED;

```

MRS <Xt>, SPSR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return SPSR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return SPSR_EL2;
elsif PSTATE.EL == EL3 then
    return SPSR_EL2;

```

MSR SPSR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        SPSR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    SPSR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    SPSR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_EL2, Saved Program Status Register (EL2)

The SPSR_EL2 characteristics are:

Purpose

Holds the saved process state when an exception is taken to EL2.

Configuration

AArch64 System register SPSR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [SPSR_hyp\[31:0\]](#).

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

SPSR_EL2 is a 64-bit register.

Field descriptions

When AArch32 is supported and exception taken from AArch32 state:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|---------|-----|------|-----|----|----|----|---------|----|----|----|----|----|------|--------|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | Z | C | V | Q | IT[1:0] | DIT | SSBS | PAN | SS | IL | GE | IT[7:2] | E | A | I | F | T | M[4] | M[3:0] | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

An exception return from EL2 using AArch64 makes SPSR_EL2 become UNKNOWN.

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL2, and copied to PSTATE.N on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL2, and copied to PSTATE.Z on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL2, and copied to PSTATE.C on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL2, and copied to PSTATE.V on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to EL2, and copied to PSTATE.Q on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to EL2, and copied to PSTATE.IT on executing an exception return operation in EL2.

SPSR_EL2.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_EL2[26:25].
- IT[7:2] is SPSR_EL2[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL2, and copied to PSTATE.DIT on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL2, and copied to PSTATE.SSBS on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]**When FEAT_PAN is implemented:**

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL2, and copied to PSTATE.PAN on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL2, and conditionally copied to PSTATE.SS on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL2, and copied to PSTATE.IL on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to EL2, and copied to PSTATE.GE on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to EL2, and copied to PSTATE.E on executing an exception return operation in EL2.

If the implementation does not support big-endian operation, SPSR_EL2.E is RES0. If the implementation does not support little-endian operation, SPSR_EL2.E is RES1. On executing an exception return operation in EL2, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_EL2.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_EL2.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL2, and copied to PSTATE.A on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL2, and copied to PSTATE.I on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL2, and copied to PSTATE.F on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to EL2, and copied to PSTATE.T on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4], bit [4]

Execution state. Set to 0b1, the value of PSTATE.nRW, on taking an exception to EL2 from AArch32 state, and copied to PSTATE.nRW on executing an exception return operation in EL2.

| M[4] | Meaning |
|-------------|--------------------------|
| 0b1 | AArch32 execution state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch32 Mode. Set to the value of PSTATE.M[3:0] on taking an exception to EL2, and copied to PSTATE.M[3:0] on executing an exception return operation in EL2.

| M[3:0] | Meaning |
|---------------|----------------|
| 0b0000 | User. |
| 0b0001 | FIQ. |
| 0b0010 | IRQ. |
| 0b0011 | Supervisor. |
| 0b0111 | Abort. |
| 0b1010 | Hyp. |
| 0b1011 | Undefined. |
| 0b1111 | System. |

Other values are reserved. If SPSR_EL2.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL2 is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When exception taken from AArch64 state:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|------|----|-----|-----|-----|----|----|------|----|----|--------|------|-------|----|----|----|----|------|------|--------|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | Z | C | V | RES0 | TC | DIT | UAO | PAN | SS | IL | RES0 | | | ALLINT | SSBS | BTYPE | D | A | I | F | RES0 | M[4] | M[3:0] | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

An exception return from EL2 using AArch64 makes SPSR_EL2 become UNKNOWN.

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL2, and copied to PSTATE.N on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL2, and copied to PSTATE.Z on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL2, and copied to PSTATE.C on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL2, and copied to PSTATE.V on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

TCO, bit [25]**When FEAT_MTE is implemented:**

Tag Check Override. Set to the value of PSTATE.TCO on taking an exception to EL2, and copied to PSTATE.TCO on executing an exception return operation in EL2.

When FEAT_MTE2 is not implemented, it is CONSTRAINED UNPREDICTABLE whether this field is RES0 or behaves as if FEAT_MTE is implemented.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [24]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL2, and copied to PSTATE.DIT on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UAO, bit [23]**When FEAT_UAO is implemented:**

User Access Override. Set to the value of PSTATE.UAO on taking an exception to EL2, and copied to PSTATE.UAO on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]**When FEAT_PAN is implemented:**

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL2, and copied to PSTATE.PAN on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL2, and conditionally copied to PSTATE.SS on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL2, and copied to PSTATE.IL on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:14]

Reserved, RES0.

ALLINT, bit [13]

When FEAT_NMI is implemented:

All IRQ or FIQ interrupts mask. Set to the value of PSTATE.ALLINT on taking an exception to EL2, and copied to PSTATE.ALLINT on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [12]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL2, and copied to PSTATE.SSBS on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BTYPE, bits [11:10]

When FEAT_BTI is implemented:

Branch Type Indicator. Set to the value of PSTATE.BTYPE on taking an exception to EL2, and copied to PSTATE.BTYPE on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D, bit [9]

Debug exception mask. Set to the value of PSTATE.D on taking an exception to EL2, and copied to PSTATE.D on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL2, and copied to PSTATE.A on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL2, and copied to PSTATE.I on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL2, and copied to PSTATE.F on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

M[4], bit [4]

Execution state. Set to 0b0, the value of PSTATE.nRW, on taking an exception to EL2 from AArch64 state, and copied to PSTATE.nRW on executing an exception return operation in EL2.

| M[4] | Meaning |
|-------------|--------------------------|
| 0b0 | AArch64 execution state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch64 Exception level and selected Stack Pointer.

| M[3:0] | Meaning |
|---------------|----------------|
| 0b0000 | EL0t. |
| 0b0100 | EL1t. |
| 0b0101 | EL1h. |
| 0b1000 | EL2t. |
| 0b1001 | EL2h. |

Other values are reserved. If SPSR_EL2.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL2 is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The bits in this field are interpreted as follows:

- M[3:2] is set to the value of PSTATE.EL on taking an exception to EL2 and copied to PSTATE.EL on executing an exception return operation in EL2.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on taking an exception to EL2 and copied to PSTATE.SP on executing an exception return operation in EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic SPSR_EL2 or SPSR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SPSR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b100 | 0b0100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return SPSR_EL1;
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return SPSR_EL2;
elsif PSTATE.EL == EL3 then
    return SPSR_EL2;

```

MSR SPSR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b100 | 0b0100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        SPSR_EL1 = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    SPSR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    SPSR_EL2 = X[t];

```

MRS <Xt>, SPSR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x160];
    else
        return SPSR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return SPSR_EL2;
    else
        return SPSR_EL1;
elsif PSTATE.EL == EL3 then
    return SPSR_EL1;

```

MSR SPSR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x160] = X[t];
    else
        SPSR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        SPSR_EL2 = X[t];
    else
        SPSR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    SPSR_EL1 = X[t];

```


SPSR_EL3, Saved Program Status Register (EL3)

The SPSR_EL3 characteristics are:

Purpose

Holds the saved process state when an exception is taken to EL3.

Configuration

AArch64 System register SPSR_EL3 bits [31:0] can be mapped to AArch32 System register [SPSR_mon\[31:0\]](#), but this is not architecturally mandated.

This register is present only when EL3 is implemented. Otherwise, direct accesses to SPSR_EL3 are UNDEFINED.

Attributes

SPSR_EL3 is a 64-bit register.

Field descriptions

When AArch32 is supported and exception taken from AArch32 state:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|---------|-----|------|-----|----|----|----|---------|----|----|----|----|----|------|--------|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | Z | C | V | Q | IT[1:0] | DIT | SSBS | PAN | SS | IL | GE | IT[7:2] | E | A | I | F | T | M[4] | M[3:0] | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

An exception return from EL3 using AArch64 makes SPSR_EL1 become UNKNOWN.

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL3, and copied to PSTATE.N on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL3, and copied to PSTATE.Z on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL3, and copied to PSTATE.C on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL3, and copied to PSTATE.V on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to EL3, and copied to PSTATE.Q on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to EL3, and copied to PSTATE.IT on executing an exception return operation in EL3.

SPSR_EL1.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_EL3[26:25].
- IT[7:2] is SPSR_EL3[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL3, and copied to PSTATE.DIT on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL3, and copied to PSTATE.SSBS on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]**When FEAT_PAN is implemented:**

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL3, and copied to PSTATE.PAN on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL3, and conditionally copied to PSTATE.SS on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL3, and copied to PSTATE.IL on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to EL3, and copied to PSTATE.GE on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to EL3, and copied to PSTATE.E on executing an exception return operation in EL3.

If the implementation does not support big-endian operation, SPSR_EL1.E is RES0. If the implementation does not support little-endian operation, SPSR_EL1.E is RES1. On executing an exception return operation in EL3, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_EL1.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_EL1.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL3, and copied to PSTATE.A on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL3, and copied to PSTATE.I on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL3, and copied to PSTATE.F on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to EL3, and copied to PSTATE.T on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4], bit [4]

Execution state. Set to 0b1, the value of PSTATE.nRW, on taking an exception to EL3 from AArch32 state, and copied to PSTATE.nRW on executing an exception return operation in EL3.

| M[4] | Meaning |
|-------------|--------------------------|
| 0b1 | AArch32 execution state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch32 Mode. Set to the value of PSTATE.M[3:0] on taking an exception to EL3, and copied to PSTATE.M[3:0] on executing an exception return operation in EL3.

| M[3:0] | Meaning |
|---------------|----------------|
| 0b0000 | User. |
| 0b0001 | FIQ. |
| 0b0010 | IRQ. |
| 0b0011 | Supervisor. |
| 0b0110 | Monitor. |
| 0b0111 | Abort. |
| 0b1010 | Hyp. |
| 0b1011 | Undefined. |
| 0b1111 | System. |

Other values are reserved. If SPSR_EL1.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL3 is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When exception taken from AArch64 state:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|------|----|-----|-----|-----|----|----|------|----|----|--------|------|-------|----|----|----|----|------|------|--------|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | Z | C | V | RES0 | TC | DIT | UAO | PAN | SS | IL | RES0 | | | ALLINT | SSBS | BTYPE | D | A | I | F | RES0 | M[4] | M[3:0] | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

An exception return from EL3 using AArch64 makes SPSR_EL1 become UNKNOWN.

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to EL3, and copied to PSTATE.N on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to EL3, and copied to PSTATE.Z on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to EL3, and copied to PSTATE.C on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to EL3, and copied to PSTATE.V on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:26]

Reserved, RES0.

TCO, bit [25]**When FEAT_MTE is implemented:**

Tag Check Override. Set to the value of PSTATE.TCO on taking an exception to EL3, and copied to PSTATE.TCO on executing an exception return operation in EL3.

When FEAT_MTE2 is not implemented, it is CONSTRAINED UNPREDICTABLE whether this field is RES0 or behaves as if FEAT_MTE is implemented.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [24]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to EL3, and copied to PSTATE.DIT on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UAO, bit [23]**When FEAT_UAO is implemented:**

User Access Override. Set to the value of PSTATE.UAO on taking an exception to EL3, and copied to PSTATE.UAO on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]**When FEAT_PAN is implemented:**

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to EL3, and copied to PSTATE.PAN on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on taking an exception to EL3, and conditionally copied to PSTATE.SS on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to EL3, and copied to PSTATE.IL on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:14]

Reserved, RES0.

ALLINT, bit [13]

When FEAT_NMI is implemented:

All IRQ or FIQ interrupts mask. Set to the value of PSTATE.ALLINT on taking an exception to EL3, and copied to PSTATE.ALLINT on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [12]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to EL3, and copied to PSTATE.SSBS on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BTYPE, bits [11:10]

When FEAT_BTI is implemented:

Branch Type Indicator. Set to the value of PSTATE.BTYPE on taking an exception to EL3, and copied to PSTATE.BTYPE on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

D, bit [9]

Debug exception mask. Set to the value of PSTATE.D on taking an exception to EL3, and copied to PSTATE.D on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to EL3, and copied to PSTATE.A on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to EL3, and copied to PSTATE.I on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to EL3, and copied to PSTATE.F on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

M[4], bit [4]

Execution state. Set to 0b0, the value of PSTATE.nRW, on taking an exception to EL3 from AArch64 state, and copied to PSTATE.nRW on executing an exception return operation in EL3.

| M[4] | Meaning |
|-------------|--------------------------|
| 0b0 | AArch64 execution state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[3:0], bits [3:0]

AArch64 Exception level and selected Stack Pointer.

| M[3:0] | Meaning |
|---------------|----------------|
| 0b0000 | EL0t. |
| 0b0100 | EL1t. |
| 0b0101 | EL1h. |
| 0b1000 | EL2t. |
| 0b1001 | EL2h. |
| 0b1100 | EL3t. |
| 0b1101 | EL3h. |

Other values are reserved. If SPSR_EL1.M[3:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in EL3 is an illegal return event, as described in 'Illegal return events from AArch64 state'.

The bits in this field are interpreted as follows:

- M[3:2] is set to the value of PSTATE.EL on taking an exception to EL3 and copied to PSTATE.EL on executing an exception return operation in EL3.
- M[1] is unused and is 0 for all non-reserved values.
- M[0] is set to the value of PSTATE.SP on taking an exception to EL3 and copied to PSTATE.SP on executing an exception return operation in EL3.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SPSR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b110 | 0b0100 | 0b0000 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SPSR_EL3;
```

MSR SPSR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b110 | 0b0100 | 0b0000 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SPSR_EL3 = X[t];
```

SPSR_fiq, Saved Program Status Register (FIQ mode)

The SPSR_fiq characteristics are:

Purpose

Holds the saved process state when an exception is taken to FIQ mode.

Configuration

AArch64 System register SPSR_fiq bits [31:0] are architecturally mapped to AArch32 System register [SPSR_fiq\[31:0\]](#).

If EL1 only supports execution in AArch64 state, this register is RES0 from EL2 and EL3.

Attributes

SPSR_fiq is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|---------|----|----|------|-----|-----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | Z | C | V | Q | IT[1:0] | | J | SSBS | PAN | DIT | IL | GE | | | | IT[7:2] | | | | E | A | I | F | T | M[4:0] | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to FIQ mode, and copied to PSTATE.N on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to FIQ mode, and copied to PSTATE.Z on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to FIQ mode, and copied to PSTATE.C on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to FIQ mode, and copied to PSTATE.V on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to FIQ mode, and copied to PSTATE.Q on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to FIQ mode, and copied to PSTATE.IT on executing an exception return operation in FIQ mode.

SPSR_fiq.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_fiq[26:25].
- IT[7:2] is SPSR_fiq[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to FIQ mode, and copied to PSTATE.SSBS on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to FIQ mode, and copied to PSTATE.PAN on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to FIQ mode, and copied to PSTATE.DIT on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to FIQ mode, and copied to PSTATE.IL on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to FIQ mode, and copied to PSTATE.GE on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to FIQ mode, and copied to PSTATE.E on executing an exception return operation in FIQ mode.

If the implementation does not support big-endian operation, SPSR_fiq.E is RES0. If the implementation does not support little-endian operation, SPSR_fiq.E is RES1. On executing an exception return operation in FIQ mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_fiq.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_fiq.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to FIQ mode, and copied to PSTATE.A on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to FIQ mode, and copied to PSTATE.I on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to FIQ mode, and copied to PSTATE.F on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to FIQ mode, and copied to PSTATE.T on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to FIQ mode, and copied to PSTATE.M[4:0] on executing an exception return operation in FIQ mode.

| M[4:0] | Meaning |
|---------------|----------------|
| 0b10000 | User. |
| 0b10001 | FIQ. |
| 0b10010 | IRQ. |
| 0b10011 | Supervisor. |
| 0b10111 | Abort. |
| 0b11011 | Undefined. |
| 0b11111 | System. |

Other values are reserved. If SPSR_fiq.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in FIQ mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_fiq

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SPSR_fiq

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b100 | 0b0100 | 0b0011 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    return SPSR_fiq;
elseif PSTATE.EL == EL3 then
    return SPSR_fiq;

```

MSR SPSR_fiq, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0100 | 0b0011 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    SPSR_fiq = X[t];
elseif PSTATE.EL == EL3 then
    SPSR_fiq = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_irq, Saved Program Status Register (IRQ mode)

The SPSR_irq characteristics are:

Purpose

Holds the saved process state when an exception is taken to IRQ mode.

Configuration

AArch64 System register SPSR_irq bits [31:0] are architecturally mapped to AArch32 System register [SPSR_irq\[31:0\]](#).

If EL1 only supports execution in AArch64 state, this register is RES0 from EL2 and EL3.

Attributes

SPSR_irq is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|---------|----|------|-----|-----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | Z | C | V | Q | IT[1:0] | J | SSBS | PAN | DIT | IL | GE | | | | IT[7:2] | | | | E | A | I | F | T | M[4:0] | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to IRQ mode, and copied to PSTATE.N on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to IRQ mode, and copied to PSTATE.Z on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to IRQ mode, and copied to PSTATE.C on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to IRQ mode, and copied to PSTATE.V on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to IRQ mode, and copied to PSTATE.Q on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to IRQ mode, and copied to PSTATE.IT on executing an exception return operation in IRQ mode.

SPSR_irq.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_irq[26:25].
- IT[7:2] is SPSR_irq[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to IRQ mode, and copied to PSTATE.SSBS on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to IRQ mode, and copied to PSTATE.PAN on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to IRQ mode, and copied to PSTATE.DIT on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to IRQ mode, and copied to PSTATE.IL on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to IRQ mode, and copied to PSTATE.GE on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to IRQ mode, and copied to PSTATE.E on executing an exception return operation in IRQ mode.

If the implementation does not support big-endian operation, SPSR_irq.E is RES0. If the implementation does not support little-endian operation, SPSR_irq.E is RES1. On executing an exception return operation in IRQ mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_irq.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_irq.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to IRQ mode, and copied to PSTATE.A on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to IRQ mode, and copied to PSTATE.I on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to IRQ mode, and copied to PSTATE.F on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to IRQ mode, and copied to PSTATE.T on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to IRQ mode, and copied to PSTATE.M[4:0] on executing an exception return operation in IRQ mode.

| M[4:0] | Meaning |
|---------------|----------------|
| 0b10000 | User. |
| 0b10001 | FIQ. |
| 0b10010 | IRQ. |
| 0b10011 | Supervisor. |
| 0b10111 | Abort. |
| 0b11011 | Undefined. |
| 0b11111 | System. |

Other values are reserved. If SPSR_irq.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in IRQ mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_irq

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SPSR_irq

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b100 | 0b0100 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    return SPSR_irq;
elseif PSTATE.EL == EL3 then
    return SPSR_irq;

```

MSR SPSR_irq, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0100 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    SPSR_irq = X[t];
elseif PSTATE.EL == EL3 then
    SPSR_irq = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_und, Saved Program Status Register (Undefined mode)

The SPSR_und characteristics are:

Purpose

Holds the saved process state when an exception is taken to Undefined mode.

Configuration

AArch64 System register SPSR_und bits [31:0] are architecturally mapped to AArch32 System register [SPSR_und\[31:0\]](#).

If EL1 only supports execution in AArch64 state, this register is RES0 from EL2 and EL3.

Attributes

SPSR_und is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|---------|----|-----|-----|-----|----|----|---------|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| N | Z | C | V | Q | IT[1:0] | J | SSB | PAN | DIT | IL | GE | IT[7:2] | E | A | I | F | T | M[4:0] | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Undefined mode, and copied to PSTATE.N on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Undefined mode, and copied to PSTATE.Z on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Undefined mode, and copied to PSTATE.C on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Undefined mode, and copied to PSTATE.V on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Undefined mode, and copied to PSTATE.Q on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Undefined mode, and copied to PSTATE.IT on executing an exception return operation in Undefined mode.

SPSR_und.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_und[26:25].
- IT[7:2] is SPSR_und[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Undefined mode, and copied to PSTATE.SSBS on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Undefined mode, and copied to PSTATE.PAN on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Undefined mode, and copied to PSTATE.DIT on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Undefined mode, and copied to PSTATE.IL on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Undefined mode, and copied to PSTATE.GE on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Undefined mode, and copied to PSTATE.E on executing an exception return operation in Undefined mode.

If the implementation does not support big-endian operation, SPSR_und.E is RES0. If the implementation does not support little-endian operation, SPSR_und.E is RES1. On executing an exception return operation in Undefined mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_und.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_und.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Undefined mode, and copied to PSTATE.A on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Undefined mode, and copied to PSTATE.I on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Undefined mode, and copied to PSTATE.F on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Undefined mode, and copied to PSTATE.T on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Undefined mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Undefined mode.

| M[4:0] | Meaning |
|---------------|----------------|
| 0b10000 | User. |
| 0b10001 | FIQ. |
| 0b10010 | IRQ. |
| 0b10011 | Supervisor. |
| 0b10111 | Abort. |
| 0b11011 | Undefined. |
| 0b11111 | System. |

Other values are reserved. If SPSR_und.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Undefined mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_und

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SPSR_und

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b100 | 0b0100 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    return SPSR_und;
elseif PSTATE.EL == EL3 then
    return SPSR_und;

```

MSR SPSR_und, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0100 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    SPSR_und = X[t];
elseif PSTATE.EL == EL3 then
    SPSR_und = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SSBS, Speculative Store Bypass Safe

The SSBS characteristics are:

Purpose

Allows access to the Speculative Store Bypass Safe bit.

Configuration

This register is present only when FEAT_SSBS is implemented. Otherwise, direct accesses to SSBS are UNDEFINED.

Attributes

SSBS is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|------|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | SSBS | | RES0 | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:13]

Reserved, RES0.

SSBS, bit [12]

Speculative Store Bypass Safe.

Prohibits speculative loads or stores which might practically allow a cache timing side channel.

A cache timing side channel might be exploited where a load or store uses an address that is derived from a register that is being loaded from memory using a load instruction speculatively read from a memory location. If PSTATE.SSBS is enabled, the address derived from the load instruction might be from earlier in the coherence order than the latest store to that memory location with the same virtual address.

| SSBS | Meaning |
|------|---|
| 0b0 | Hardware is not permitted to load or store speculatively, in a manner that could practically give rise to a cache timing side channel, using an address derived from a register value that has been loaded from memory using a load instruction (L) that speculatively reads an entry from earlier in the coherence order from that location being loaded from than the entry generated by the latest store (S) to that location using the same virtual address as L. |
| 0b1 | Hardware is permitted to load or store speculatively, in a manner that could practically give rise to a cache timing side channel, using an address derived from a register value that has been loaded from memory using a load instruction (L) that speculatively reads an entry from earlier in the coherence order from that location being loaded from than the entry generated by the latest store (S) to that location using the same virtual address as L. |

The value of this bit is set to the value in the SCTLRL_ELx.DSSBS field on taking an exception to ELx.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Bits [11:0]

Reserved, RES0.

Accessing SSBS

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SSBS

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    return Zeros(51):PSTATE.SSBS:Zeros(12);
elsif PSTATE.EL == EL1 then
    return Zeros(51):PSTATE.SSBS:Zeros(12);
elsif PSTATE.EL == EL2 then
    return Zeros(51):PSTATE.SSBS:Zeros(12);
elsif PSTATE.EL == EL3 then
    return Zeros(51):PSTATE.SSBS:Zeros(12);

```

MSR SSBS, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    PSTATE.SSBS = X[t]<12>;
elsif PSTATE.EL == EL1 then
    PSTATE.SSBS = X[t]<12>;
elsif PSTATE.EL == EL2 then
    PSTATE.SSBS = X[t]<12>;
elsif PSTATE.EL == EL3 then
    PSTATE.SSBS = X[t]<12>;

```

MSR SSBS, #<imm>

| op0 | op1 | CRn | op2 |
|------|-------|--------|-------|
| 0b00 | 0b011 | 0b0100 | 0b001 |

SVCR, Streaming Vector Control Register

The SVCR characteristics are:

Purpose

Controls Streaming SVE mode and SME behavior.

Configuration

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to SVCR are UNDEFINED.

Attributes

SVCR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ZA |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | SM |

Bits [63:2]

Reserved, RES0.

ZA, bit [1]

Enables ZA array storage. The possible values of this bit are:

| ZA | Meaning |
|-----|---|
| 0b0 | SME ZA array storage is invalid and not accessible. SME instructions that access the ZA array are illegal. |
| 0b1 | SME ZA array storage is valid and accessible. If SME instructions are not trapped, SME instructions that access the ZA array are legal. |

When the value of PSTATE.ZA is changed, the following applies:

- When changed from 0 to 1, all implemented bits of the SME ZA storage are set to zero.
- When changed from 1 to 0, there is no observable change to SME ZA storage.
- There is no effect on the SVE vector and predicate registers and [FPSR](#).

A direct or indirect read of ZA appears to occur in program order relative to a direct write of SVCR, and to MSR SVCRZA and MSR SVCRSMZA instructions, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

SM, bit [0]

Enables Streaming SVE mode. The possible values of this bit are:

| SM | Meaning |
|-----|--------------------------------------|
| 0b0 | The PE is not in Streaming SVE mode. |
| 0b1 | The PE is in Streaming SVE mode. |

When the value of PSTATE.SM is changed, the following applies:

- When changed from 0 to 1, an entry to Streaming SVE mode is performed.
- When changed from 1 to 0, an exit from Streaming SVE mode is performed.
- All implemented bits of the SVE registers Z0-Z31, P0-P15, and FFR in the new mode are set to zero.
- [FPSR](#) in the new mode is set to 0x0000_0000_0800_009f, in which all cumulative status bits are set to 1.
- There is no effect on ZA storage.

A direct or indirect read of SM appears to occur in program order relative to a direct write of SVCR, and to MSR SVCRSM and MSR SVCRSMZA instructions, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing SVCR

SVCR is read/write and can be accessed from any Exception level.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, SVCR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CPACR_EL1.SMEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        else
            AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.SMEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        return Zeros(62):PSTATE.<ZA,SM>;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        return Zeros(62):PSTATE.<ZA,SM>;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        return Zeros(62):PSTATE.<ZA,SM>;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        return Zeros(62):PSTATE.<ZA,SM>;

```

MSR SVCR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CPACR_EL1.SMEN != '11' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x1D);
        else
            AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.SMEN != '11' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        bits(64) v = X[t]; SetPSTATE_SM(v<0>); SetPSTATE_ZA(v<1>);
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif CPACR_EL1.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        bits(64) v = X[t]; SetPSTATE_SM(v<0>); SetPSTATE_ZA(v<1>);
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.ESM == '0' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TSM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.SMEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x1D);
    elsif HaveEL(EL3) && CPTR_EL3.ESM == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        bits(64) v = X[t]; SetPSTATE_SM(v<0>); SetPSTATE_ZA(v<1>);
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.ESM == '0' then
        AArch64.SystemAccessTrap(EL3, 0x1D);
    else
        bits(64) v = X[t]; SetPSTATE_SM(v<0>); SetPSTATE_ZA(v<1>);

```

MSR SVCRSM, #<imm>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b00 | 0b011 | 0b0100 | 0b001x | 0b011 |

MSR SVCRZA, #<imm>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b00 | 0b011 | 0b0100 | 0b010x | 0b011 |

MSR SVCRCMZA, #<imm>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b00 | 0b011 | 0b0100 | 0b011x | 0b011 |

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCO, Tag Check Override

The TCO characteristics are:

Purpose

When FEAT_MTE is implemented, this register allows tag checks to be disabled globally.

When FEAT_MTE2 is not implemented, it is CONstrained UNpredictable whether this register is RES0 or behaves as if FEAT_MTE is implemented.

Configuration

This register is present only when FEAT_MTE is implemented. Otherwise, direct accesses to TCO are UNDEFINED.

Attributes

TCO is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|-----|----|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | TCO | | RES0 | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:26]

Reserved, RES0.

TCO, bit [25]

Allows memory tag checks to be globally disabled.

| TCO | Meaning |
|-----|--|
| 0b0 | Loads and Stores are not affected by this control. |
| 0b1 | Loads and Stores are unchecked. |

Bits [24:0]

Reserved, RES0.

Accessing TCO

For information about the operation of the MSR (immediate) accessor, see MSR (immediate).

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TCO

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b111 |

```

if PSTATE.EL == EL0 then
    return Zeros(38):PSTATE.TC0:Zeros(25);
elsif PSTATE.EL == EL1 then
    return Zeros(38):PSTATE.TC0:Zeros(25);
elsif PSTATE.EL == EL2 then
    return Zeros(38):PSTATE.TC0:Zeros(25);
elsif PSTATE.EL == EL3 then
    return Zeros(38):PSTATE.TC0:Zeros(25);

```

MSR TC0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b111 |

```

if PSTATE.EL == EL0 then
    PSTATE.TC0 = X[t]<25>;
elsif PSTATE.EL == EL1 then
    PSTATE.TC0 = X[t]<25>;
elsif PSTATE.EL == EL2 then
    PSTATE.TC0 = X[t]<25>;
elsif PSTATE.EL == EL3 then
    PSTATE.TC0 = X[t]<25>;

```

MSR TC0, #<imm>

| op0 | op1 | CRn | op2 |
|------|-------|--------|-------|
| 0b00 | 0b011 | 0b0100 | 0b100 |

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCR_EL1, Translation Control Register (EL1)

The TCR_EL1 characteristics are:

Purpose

The control register for stage 1 of the EL1&0 translation regime.

Configuration

AArch64 System register TCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [TTBCR\[31:0\]](#).

AArch64 System register TCR_EL1 bits [63:32] are architecturally mapped to AArch32 System register [TTBCR2\[31:0\]](#).

Attributes

TCR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | |
|------|-----|-------|-------|-------|-------|------|------|-------|-------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 |
| RES0 | DS | TCMA1 | TCMA0 | EOPD1 | EOPD0 | NFD1 | NFD0 | TBID1 | TBID0 | HWU162 | HWU161 | HWU160 | HWU159 | HWU158 | HWU157 | HWU156 | HWU155 | HWU154 |
| TG1 | SH1 | ORGN1 | IRGN1 | EPD1 | A1 | T1SZ | | | | | | | | | TG0 | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 |

Any of the bits in TCR_EL1, other than the A1 bit and the EPDx bits when they have the value 1, are permitted to be cached in a TLB.

Bits [63:60]

Reserved, RES0.

DS, bit [59]

When FEAT_LPA2 is implemented:

This field affects 52-bit output addressing when using 4KB and 16KB translation granules in stage 1 of the EL1&0 translation regime.

| DS | Meaning |
|-----|---|
| 0b0 | <p>Bits[49:48] of translation descriptors are RES0.</p> <p>Bits[9:8] in block and page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in table descriptors are ignored by hardware.</p> <p>The minimum value of the TCR_EL1.{T0SZ, T1SZ} fields is 16. Any memory access using a smaller value generates a stage 1 level 0 translation table fault.</p> <p>Output address[51:48] is 0b0000.</p> |
| 0b1 | <p>Bits[49:48] of translation descriptors hold output address[49:48].</p> <p>Bits[9:8] of translation table descriptors hold output address[51:50].</p> <p>The shareability information of block and page descriptors for cacheable locations is determined by:</p> <ul style="list-style-type: none"> • TCR_EL1.SH0 if the VA is translated using tables pointed to by TTBR0_EL1. • TCR_EL1.SH1 if the VA is translated using tables pointed to by TTBR1_EL1. <p>The minimum value of the TCR_EL1.{T0SZ, T1SZ} fields is 12. Any memory access using a smaller value generates a stage 1 level 0 translation table fault.</p> <p>All calculations of the stage 1 base address are modified for tables of fewer than 8 entries so that the table is aligned to 64 bytes.</p> <p>Bits[5:2] of TTBR0_EL1 or TTBR1_EL1 are used to hold bits[51:48] of the output address in all cases.</p> <hr/> <p>Note</p> <p>As FEAT_LVA must be implemented if TCR_EL1.DS == 1, the minimum value of the TCR_EL1.{T0SZ, T1SZ} fields is 12, as determined by that extension.</p> <hr/> <p>For the TLBI Range instructions affecting VA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0b0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 0b00.</p> <hr/> <p>Note</p> <p>This forces alignment of the ranges used by the TLBI range instructions.</p> <hr/> |

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA1, bit [58]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL1, and at EL0 if [HCR_EL2](#).{E2H,TGE}!={1,1}, when address[59:55] = 0b11111.

| TCMA1 | Meaning |
|-------------|---|
| 0b0 | This control has no effect on the generation of Unchecked accesses at EL1 or EL0. |
| 0b1 | All accesses at EL1 and EL0 are Unchecked. |
| Note | |

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA0, bit [57]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL1, and at EL0 if [HCR_EL2](#).{E2H,TGE}!= {1,1}, when address[59:55] = 0b00000.

| TCMA0 | Meaning |
|-------|---|
| 0b0 | This control has no effect on the generation of Unchecked accesses at EL1 or EL0. |
| 0b1 | All accesses at EL1 and EL0 are Unchecked. |

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EOPD1, bit [56]

When FEAT_EOPD is implemented:

Faulting control for Unprivileged access to any address translated by [TTBR1_EL1](#).

| EOPD1 | Meaning |
|-------|---|
| 0b0 | Unprivileged access to any address translated by TTBR1_EL1 will not generate a fault by this mechanism. |
| 0b1 | Unprivileged access to any address translated by TTBR1_EL1 will generate a level 0 Translation fault. |

Level 0 Translation faults generated as a result of this field are not counted as TLB misses for performance monitoring. The fault should take the same time to generate, whether the address is present in the TLB or not, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EOPD0, bit [55]**When FEAT_EOPD is implemented:**

Faulting control for Unprivileged access to any address translated by [TTBR0_EL1](#).

| EOPD0 | Meaning |
|--------------|---|
| 0b0 | Unprivileged access to any address translated by TTBR0_EL1 will not generate a fault by this mechanism. |
| 0b1 | Unprivileged access to any address translated by TTBR0_EL1 will generate a level 0 Translation fault. |

Level 0 Translation faults generated as a result of this field are not counted as TLB misses for performance monitoring. The fault should take the same time to generate, whether the address is present in the TLB or not, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD1, bit [54]**When FEAT_SVE is implemented or FEAT_TME is implemented:**

Non-fault translation table walk disable for stage 1 translations using [TTBR1_EL1](#).

This bit controls whether to perform a stage 1 translation table walk in response to a non-fault unprivileged access for a virtual address that is translated using [TTBR1_EL1](#).

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

For more information, see 'The Scalable Vector Extension (SVE)'.

If FEAT_TME is implemented, the affected access types include all accesses generated by a load or store instruction in Transactional state.

| NFD1 | Meaning |
|-------------|---|
| 0b0 | Does not disable stage 1 translation table walks using TTBR1_EL1 . |
| 0b1 | A TLB miss on a virtual address that is translated using TTBR1_EL1 due to the specified access types causes the access to fail without taking an exception. No stage 1 translation table walk is performed. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD0, bit [53]**When FEAT_SVE is implemented or FEAT_TME is implemented:**

Non-fault translation table walk disable for stage 1 translations using [TTBR0_EL1](#).

This bit controls whether to perform a stage 1 translation table walk in response to a non-fault unprivileged access for a virtual address that is translated using [TTBR0_EL1](#).

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

For more information, see 'The Scalable Vector Extension (SVE)'.

If FEAT_TME is implemented, the affected access types include all accesses generated by a load or store instruction in Transactional state.

| NFD0 | Meaning |
|------|---|
| 0b0 | Does not disable stage 1 translation table walks using TTBR0_EL1 . |
| 0b1 | A TLB miss on a virtual address that is translated using TTBR0_EL1 due to the specified access types causes the access to fail without taking an exception. No stage 1 translation table walk is performed. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID1, bit [52]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see 'Address tagging in AArch64 state'.

| TBID1 | Meaning |
|-------|--|
| 0b0 | TCR_EL1.TBI1 applies to Instruction and Data accesses. |
| 0b1 | TCR_EL1.TBI1 applies to Data accesses only. |

This affects addresses where the address would be translated by tables pointed to by [TTBR1_EL1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID0, bit [51]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see 'Address tagging in AArch64 state'.

| TBID0 | Meaning |
|--------------|--|
| 0b0 | TCR_EL1.TBI0 applies to Instruction and Data accesses. |
| 0b1 | TCR_EL1.TBI0 applies to Data accesses only. |

This affects addresses where the address would be translated by tables pointed to by [TTBR0_EL1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU162, bit [50]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using [TTBR1_EL1](#).

| HWU162 | Meaning |
|---------------|---|
| 0b0 | For translations using TTBR1_EL1 , bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR1_EL1 , bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1. |

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU161, bit [49]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using [TTBR1_EL1](#).

| HWU161 | Meaning |
|---------------|---|
| 0b0 | For translations using TTBR1_EL1 , bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR1_EL1 , bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1. |

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU160, bit [48]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using [TTBR1_EL1](#).

| HWU160 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR1_EL1 , bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR1_EL1 , bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1. |

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU159, bit [47]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using [TTBR1_EL1](#).

| HWU159 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR1_EL1 , bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR1_EL1 , bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD1 is 1. |

The Effective value of this field is 0 if the value of TCR_EL1.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU062, bit [46]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using [TTBR0_EL1](#).

| HWU062 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR0_EL1 , bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR0_EL1 , bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1. |

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU061, bit [45]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using [TTBR0_EL1](#).

| HWU061 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR0_EL1 , bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR0_EL1 , bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1. |

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU060, bit [44]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using [TTBR0_EL1](#).

| HWU060 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR0_EL1 , bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR0_EL1 , bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1. |

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU059, bit [43]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using [TTBR0_EL1](#).

| HWU059 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR0_EL1 , bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR0_EL1 , bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL1.HPD0 is 1. |

The Effective value of this field is 0 if the value of TCR_EL1.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HPD1, bit [42]**When FEAT_HPDS is implemented:**

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by [TTBR1_EL1](#).

| HPD1 | Meaning |
|------|--|
| 0b0 | Hierarchical permissions are enabled. |
| 0b1 | Hierarchical permissions are disabled. |

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD0, bit [41]**When FEAT_HPDS is implemented:**

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by [TTBR0_EL1](#).

| HPD0 | Meaning |
|------|--|
| 0b0 | Hierarchical permissions are enabled. |
| 0b1 | Hierarchical permissions are disabled. |

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HD, bit [40]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 1 translations from EL0 and EL1.

| HD | Meaning |
|-----|--|
| 0b0 | Stage 1 hardware management of dirty state disabled. |
| 0b1 | Stage 1 hardware management of dirty state enabled, only if the HA bit is also set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [39]

When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 1 translations from EL0 and EL1.

| HA | Meaning |
|-----|--------------------------------------|
| 0b0 | Stage 1 Access flag update disabled. |
| 0b1 | Stage 1 Access flag update enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI1, bit [38]

Top Byte ignored. Indicates whether the top byte of an address is used for address match for the [TTBR1_EL1](#) region, or ignored and used for tagged addresses.

| TBI1 | Meaning |
|------|--|
| 0b0 | Top Byte used in the address calculation. |
| 0b1 | Top Byte ignored in the address calculation. |

This affects addresses generated in EL0 and EL1 using AArch64 where the address would be translated by tables pointed to by [TTBR1_EL1](#). It has an effect whether the EL1&0 translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL1.TBID1 is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI1 is 1 and bit [55] of the target address to be stored to the PC is 1, then bits[63:56] of that target address are also set to 1 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TBIO, bit [37]

Top Byte ignored. Indicates whether the top byte of an address is used for address match for the [TTBR0_EL1](#) region, or ignored and used for tagged addresses.

| TBIO | Meaning |
|------|--|
| 0b0 | Top Byte used in the address calculation. |
| 0b1 | Top Byte ignored in the address calculation. |

This affects addresses generated in EL0 and EL1 using AArch64 where the address would be translated by tables pointed to by [TTBR0_EL1](#). It has an effect whether the EL1&0 translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL1.TBID0 is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBIO is 1 and bit [55] of the target address to be stored to the PC is 0, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AS, bit [36]

ASID Size.

| AS | Meaning |
|-----|--|
| 0b0 | 8 bit - the upper 8 bits of TTBR0_EL1 and TTBR1_EL1 are ignored by hardware for every purpose except reading back the register, and are treated as if they are all zeros for when used for allocation and matching entries in the TLB. |
| 0b1 | 16 bit - the upper 16 bits of TTBR0_EL1 and TTBR1_EL1 are used for allocation and matching in the TLB. |

If the implementation has only 8 bits of ASID, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [35]

Reserved, RES0.

IPS, bits [34:32]

Intermediate Physical Address Size.

| IPS | Meaning |
|-------|-----------------|
| 0b000 | 32 bits, 4GB. |
| 0b001 | 36 bits, 64GB. |
| 0b010 | 40 bits, 1TB. |
| 0b011 | 42 bits, 4TB. |
| 0b100 | 44 bits, 16TB. |
| 0b101 | 48 bits, 256TB. |
| 0b110 | 52 bits, 4PB. |

All other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

If the translation granule is not 64KB and FEAT_LPA2 is not implemented, the value 0b110 is treated as reserved.

It is IMPLEMENTATION DEFINED whether an implementation that does not implement FEAT_LPA supports setting the value of 0b110 for the 64KB translation granule size or whether setting this value behaves as the 0b101 encoding.

In an implementation that supports 52-bit PAs, if the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by TCR_EL1 are 0b0000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG1, bits [31:30]

Granule size for the [TTBR1_EL1](#).

| TG1 | Meaning |
|------|---------|
| 0b01 | 16KB. |
| 0b10 | 4KB. |
| 0b11 | 64KB. |

Other values are reserved.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH1, bits [29:28]

Shareability attribute for memory associated with translation table walks using [TTBR1_EL1](#).

| SH1 | Meaning |
|------|------------------|
| 0b00 | Non-shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN1, bits [27:26]

Outer cacheability attribute for memory associated with translation table walks using [TTBR1_EL1](#).

| ORGN1 | Meaning |
|-------|---|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN1, bits [25:24]

Inner cacheability attribute for memory associated with translation table walks using [TTBR1_EL1](#).

| IRGN1 | Meaning |
|-------|---|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EPD1, bit [23]

Translation table walk disable for translations using [TTBR1_EL1](#). This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using [TTBR1_EL1](#). The encoding of this bit is:

| EPD1 | Meaning |
|------|--|
| 0b0 | Perform translation table walks using TTBR1_EL1 . |
| 0b1 | A TLB miss on an address that is translated using TTBR1_EL1 generates a Translation fault. No translation table walk is performed. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A1, bit [22]

Selects whether [TTBR0_EL1](#) or [TTBR1_EL1](#) defines the ASID. The encoding of this bit is:

| A1 | Meaning |
|-----|---|
| 0b0 | TTBR0_EL1 .ASID defines the ASID. |
| 0b1 | TTBR1_EL1 .ASID defines the ASID. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T1SZ, bits [21:16]

The size offset of the memory region addressed by [TTBR1_EL1](#). The region size is $2^{(64-T1SZ)}$ bytes.

The maximum and minimum possible values for T1SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the [TTBR0_EL1](#).

| TG0 | Meaning |
|------------|----------------|
| 0b00 | 4KB |
| 0b01 | 64KB |
| 0b10 | 16KB |

Other values are reserved.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using [TTBR0_EL1](#).

| SH0 | Meaning |
|------------|-----------------|
| 0b00 | Non-shareable |
| 0b10 | Outer Shareable |
| 0b11 | Inner Shareable |

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using [TTBR0_EL1](#).

| ORGN0 | Meaning |
|--------------|---|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using [TTBR0_EL1](#).

| IRGN0 | Meaning |
|-------|---|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EPD0, bit [7]

Translation table walk disable for translations using [TTBR0_EL1](#). This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using [TTBR0_EL1](#). The encoding of this bit is:

| EPD0 | Meaning |
|------|--|
| 0b0 | Perform translation table walks using TTBR0_EL1 . |
| 0b1 | A TLB miss on an address that is translated using TTBR0_EL1 generates a Translation fault. No translation table walk is performed. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by [TTBR0_EL1](#). The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TCR_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic TCR_EL1 or TCR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b010 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x120];
    else
        return TCR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TCR_EL2;
    else
        return TCR_EL1;
elsif PSTATE.EL == EL3 then
    return TCR_EL1;

```

MSR TCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x120] = X[t];
    else
        TCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        TCR_EL2 = X[t];
    else
        TCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TCR_EL1 = X[t];

```

MRS <Xt>, TCR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0010 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x120];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return TCR_EL1;
    else
        UNDEFINED;

```

MSR TCR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0010 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x120] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        TCR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        TCR_EL1 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCR_EL2, Translation Control Register (EL2)

The TCR_EL2 characteristics are:

Purpose

The control register for stage 1 of the EL2, or EL2&0, translation regime:

- When the Effective value of [HCR_EL2.E2H](#) is 0, this register controls stage 1 of the EL2 translation regime, that supports a single VA range, translated using [TTBR0_EL2](#).
- When the value of [HCR_EL2.E2H](#) is 1, this register controls stage 1 of the EL2&0 translation regime, that supports both:
 - A lower VA range, translated using [TTBR0_EL2](#).
 - A higher VA range, translated using [TTBR1_EL2](#).

Configuration

AArch64 System register TCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HTCR\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

TCR_EL2 is a 64-bit register.

Field descriptions

When HCR_EL2.E2H == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|------|-------|-------|-------|-------|-----|------|----|----|----|------|----|-----|-----|-------|-------|------|------|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES1 | TCMA | TBID | HWU62 | HWU61 | HWU60 | HWU59 | HPD | RES1 | HD | HA | TB | RES0 | PS | TG0 | SH0 | ORGN0 | IRGN0 | RES0 | T0SZ | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | |

Any of the bits in TCR_EL2, other than the A1 bit and the EPDx bits when they have the value 1, are permitted to be cached in a TLB.

Bits [63:33]

Reserved, RES0.

DS, bit [32]

When FEAT_LPA2 is implemented:

This field affects 52-bit output addressing when using 4KB and 16KB translation granules in stage 1 of the EL2 translation regime.

| DS | Meaning |
|-----|---|
| 0b0 | <p>Bits[49:48] of translation descriptors are RES0.</p> <p>Bits[9:8] in block and page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in table descriptors are ignored by hardware.</p> <p>The minimum value of TCR_EL2.T0SZ is 16. Any memory access using a smaller value generates a stage 1 level 0 translation table fault.</p> <p>Output address[51:48] is 0b0000.</p> |
| 0b1 | <p>Bits[49:48] of translation descriptors hold output address[49:48].</p> <p>Bits[9:8] of translation table descriptors hold output address[51:50].</p> <p>The shareability information of block and page descriptors for cacheable locations is determined by TCR_EL2.SH0.</p> <p>The minimum value of TCR_EL2.T0SZ is 12. Any memory access using a smaller value generates a stage 1 level 0 translation table fault.</p> <p>All calculations of the stage 1 base address are modified for tables of fewer than 8 entries so that the table is aligned to 64 bytes.</p> <p>Bits[5:2] of TTBR0_EL2 are used to hold bits[51:48] of the output address in all cases.</p> <hr/> <p>Note</p> <p>As FEAT_LVA must be implemented if TCR_EL2.DS == 1, the minimum value of the TCR_EL2.T0SZ field is 12, as determined by that extension.</p> <hr/> <p>For the TLBI Range instructions affecting VA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0b0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 0b00.</p> <hr/> <p>Note</p> <p>This forces alignment of the ranges used by the TLBI range instructions.</p> <hr/> |

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [31]

Reserved, RES1.

TCMA, bit [30]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL2 when address [59:56] = 0b0000.

| TCMA | Meaning |
|------|---|
| 0b0 | This control has no effect on the generation of Unchecked accesses. |
| 0b1 | All accesses are Unchecked. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID, bit [29]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see 'Address tagging in AArch64 state'.

| TBID | Meaning |
|------|---|
| 0b0 | TCR_EL2.TBI applies to Instruction and Data accesses. |
| 0b1 | TCR_EL2.TBI applies to Data accesses only. |

This affects addresses where the address would be translated by tables pointed to by [TTBR0_EL2](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry.

| HWU62 | Meaning |
|-------|---|
| 0b0 | Bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU61, bit [27]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry.

| HWU61 | Meaning |
|-------|---|
| 0b0 | Bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU60, bit [26]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry.

| HWU60 | Meaning |
|-------|---|
| 0b0 | Bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU59, bit [25]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry.

| HWU59 | Meaning |
|-------|---|
| 0b0 | Bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HPD, bit [24]

When FEAT_HPDS is implemented:

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by [TTBR0_EL2](#).

| HPD | Meaning |
|-----|--|
| 0b0 | Hierarchical permissions are enabled. |
| 0b1 | Hierarchical permissions are disabled. |

Note

In this case, bit[61] (APTable[0]) and bit[59] (PXNTable) of the next level descriptor attributes are required to be ignored by the PE and are no longer reserved, allowing them to be used by software.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES1.

HD, bit [22]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 1 translations from EL2.

| HD | Meaning |
|-----|--|
| 0b0 | Stage 1 hardware management of dirty state disabled. |
| 0b1 | Stage 1 hardware management of dirty state enabled, only if the HA bit is also set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]

When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 1 translations from EL2.

| HA | Meaning |
|-----|--------------------------------------|
| 0b0 | Stage 1 Access flag update disabled. |
| 0b1 | Stage 1 Access flag update enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI, bit [20]

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the [TTBR0_EL2](#) region, or ignored and used for tagged addresses.

For more information, see 'Address tagging in AArch64 state'.

| TBI | Meaning |
|-----|--|
| 0b0 | Top Byte used in the address calculation. |
| 0b1 | Top Byte ignored in the address calculation. |

This affects addresses generated in EL2 using AArch64 where the address would be translated by tables pointed to by [TTBR0_EL2](#). It has an effect whether the EL2, or EL2&0, translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL2.TBID is 1, then this field only applies to Data accesses.

If the value of TBI is 1, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL2.
- An exception taken to EL2.
- An exception return to EL2.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

PS, bits [18:16]

Physical Address Size.

| PS | Meaning |
|-------|-----------------|
| 0b000 | 32 bits, 4GB. |
| 0b001 | 36 bits, 64GB. |
| 0b010 | 40 bits, 1TB. |
| 0b011 | 42 bits, 4TB. |
| 0b100 | 44 bits, 16TB. |
| 0b101 | 48 bits, 256TB. |
| 0b110 | 52 bits, 4PB. |

All other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

If the translation granule is not 64KB and FEAT_LPA2 is not implemented, the value 0b110 is treated as reserved.

It is IMPLEMENTATION DEFINED whether an implementation that does not implement FEAT_LPA supports setting the value of 0b110 for the 64KB translation granule size or whether setting this value behaves as the 0b101 encoding.

In an implementation that supports 52-bit PAs, if the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by TCR_EL2 are 0b0000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the [TTBR0_EL2](#).

| TG0 | Meaning |
|------|---------|
| 0b00 | 4KB. |
| 0b01 | 64KB. |
| 0b10 | 16KB. |

Other values are reserved.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using [TTBR0_EL2](#).

| SH0 | Meaning |
|------|------------------|
| 0b00 | Non-shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using [TTBR0_EL2](#).

| ORGN0 | Meaning |
|-------|---|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using [TTBR0_EL2](#).

| IRGN0 | Meaning |
|-------|---|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:6]

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by [TTBR0_EL2](#). The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When FEAT_VHE is implemented and HCR_EL2.E2H == 1:

| | | | | | | | | | | | | | | | | | | |
|------|-----|-------|-------|-------|-------|------|------|-------|-------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 |
| RES0 | DS | TCMA1 | TCMA0 | EOPD1 | EOPD0 | NFD1 | NFD0 | TBID1 | TBID0 | HWU162 | HWU161 | HWU160 | HWU159 | HWU158 | HWU157 | HWU156 | HWU155 | HWU154 |
| TG1 | SH1 | ORGN1 | IRGN1 | EPD1 | A1 | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 |

Any of the bits in TCR_EL2 are permitted to be cached in a TLB.

Bits [63:60]

Reserved, RES0.

DS, bit [59]

When FEAT_LPA2 is implemented:

This field affects 52-bit output addressing when using 4KB and 16KB translation granules in stage 1 of the EL2&0 translation regime.

| DS | Meaning |
|-----|--|
| 0b0 | <p>Bits[49:48] of translation descriptors are RES0.</p> <p>Bits[9:8] in block and page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in table descriptors are ignored by hardware.</p> <p>The minimum value of the TCR_EL2.{T0SZ, T1SZ} fields is 16. Any memory access using a smaller value generates a stage 1 level 0 translation table fault.</p> <p>Output address[51:48] is 0b0000.</p> |
| 0b1 | <p>Bits[49:48] of translation descriptors hold output address[49:48].</p> <p>Bits[9:8] of translation table descriptors hold output address[51:50].</p> <p>The shareability information of block and page descriptors for cacheable locations is determined by:</p> <ul style="list-style-type: none"> • TCR_EL2.SH0 if the VA is an address that is translated using tables pointed to by TTBR0_EL2. • TCR_EL2.SH1 if the VA is an address that is translated using tables pointed to by TTBR1_EL2. <p>The minimum value of the TCR_EL2.{T0SZ, T1SZ} fields is 12. Any memory access using a smaller value generates a stage 1 level 0 translation table fault.</p> <p>All calculations of the stage 1 base address are modified for tables of fewer than 16 entries so that the table is aligned to 64 bytes.</p> <p>Bits[5:2] of TTBR0_EL2 or TTBR1_EL2 are used to hold bits[51:48] of the output address in all cases.</p> <hr/> <p>Note</p> <p>As FEAT_LVA must be implemented if TCR_EL2.DS == 1, the minimum value of the TCR_EL2.{T0SZ, T1SZ} fields is 12, as determined by that extension.</p> <hr/> <p>For the TLBI Range instructions affecting VA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0b0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 0b00.</p> <hr/> <p>Note</p> <p>This forces alignment of the ranges used by the TLBI range instructions.</p> <hr/> |

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA1, bit [58]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL2, and at EL0 if HCR_EL2.TGE=1, when address[59:55] = 0b11111.

| TCMA1 | Meaning |
|-------------|---|
| 0b0 | This control has no effect on the generation of Unchecked accesses at EL2 or EL0. |
| 0b1 | All accesses are Unchecked. |
| Note | |

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TCMA0, bit [57]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL2, and at EL0 if HCR_EL2.TGE=1, when address[59:55] = 0b00000.

| TCMA0 | Meaning |
|-------|---|
| 0b0 | This control has no effect on the generation of Unchecked accesses at EL2 or EL0. |
| 0b1 | All accesses are Unchecked. |

Note

Software may change this control bit on a context switch.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EOPD1, bit [56]

When FEAT_EOPD is implemented:

Faulting control for Unprivileged access to any address translated by [TTBR1_EL2](#).

| EOPD1 | Meaning |
|-------|---|
| 0b0 | Unprivileged access to any address translated by TTBR1_EL2 will not generate a fault by this mechanism. |
| 0b1 | Unprivileged access to any address translated by TTBR1_EL2 will generate a level 0 Translation fault. |

Level 0 Translation faults generated as a result of this field are not counted as TLB misses for performance monitoring. The fault should take the same time to generate, whether the address is present in the TLB or not, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EOPD0, bit [55]**When FEAT_EOPD is implemented:**

Faulting control for Unprivileged access to any address translated by [TTBR0_EL2](#).

| EOPD0 | Meaning |
|--------------|---|
| 0b0 | Unprivileged access to any address translated by TTBR0_EL2 will not generate a fault by this mechanism. |
| 0b1 | Unprivileged access to any address translated by TTBR0_EL2 will generate a level 0 Translation fault. |

Level 0 Translation faults generated as a result of this field are not counted as TLB misses for performance monitoring. The fault should take the same time to generate, whether the address is present in the TLB or not, to mitigate attacks that use fault timing.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD1, bit [54]**When FEAT_SVE is implemented or FEAT_TME is implemented:**

Non-fault translation table walk disable for stage 1 translations using [TTBR1_EL2](#).

This bit controls whether to perform a stage 1 translation table walk in response to a non-fault unprivileged access for a virtual address that is translated using [TTBR1_EL2](#).

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

For more information, see 'The Scalable Vector Extension (SVE)'.

If FEAT_TME is implemented, the affected access types include all accesses generated by a load or store instruction in Transactional state.

| NFD1 | Meaning |
|-------------|---|
| 0b0 | Does not disable stage 1 translation table walks using TTBR1_EL2 . |
| 0b1 | A TLB miss on a virtual address that is translated using TTBR1_EL2 due to the specified access types causes the access to fail without taking an exception. No stage 1 translation table walk is performed. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NFD0, bit [53]**When FEAT_SVE is implemented or FEAT_TME is implemented:**

Non-fault translation table walk disable for stage 1 translations using [TTBR0_EL2](#).

This bit controls whether to perform a stage 1 translation table walk in response to a non-fault unprivileged access for a virtual address that is translated using [TTBR0_EL2](#).

If SVE is implemented, the affected access types include:

- All accesses due to an SVE non-fault contiguous load instruction.
- Accesses due to an SVE first-fault gather load instruction that are not for the First active element. Accesses due to an SVE first-fault contiguous load instruction are not affected.
- Accesses due to prefetch instructions might be affected, but the effect is not architecturally visible.

For more information, see 'The Scalable Vector Extension (SVE)'.

If FEAT_TME is implemented, the affected access types include all accesses generated by a load or store instruction in Transactional state.

| NFD0 | Meaning |
|------|---|
| 0b0 | Does not disable stage 1 translation table walks using TTBR0_EL2 . |
| 0b1 | A TLB miss on a virtual address that is translated using TTBR0_EL2 due to the specified access types causes the access to fail without taking an exception. No stage 1 translation table walk is performed. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID1, bit [52]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

For more information, see 'Address tagging in AArch64 state'.

| TBID1 | Meaning |
|-------|--|
| 0b0 | TCR_EL2.TBI1 applies to Instruction and Data accesses. |
| 0b1 | TCR_EL2.TBI1 applies to Data accesses only. |

This affects addresses where the address would be translated by tables pointed to by [TTBR1_EL2](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID0, bit [51]

When FEAT_PAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

For more information, see 'Address tagging in AArch64 state'.

| TBID0 | Meaning |
|-------|--|
| 0b0 | TCR_EL2.TBI0 applies to Instruction and Data accesses. |
| 0b1 | TCR_EL2.TBI0 applies to Data accesses only. |

This affects addresses where the address would be translated by tables pointed to by [TTBR0_EL2](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU162, bit [50]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using [TTBR1_EL2](#).

| HWU162 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR1_EL2 , bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR1_EL2 , bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU161, bit [49]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using [TTBR1_EL2](#).

| HWU161 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR1_EL2 , bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR1_EL2 , bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU160, bit [48]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using [TTBR1_EL2](#).

| HWU160 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR1_EL2 , bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR1_EL2 , bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU159, bit [47]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using [TTBR1_EL2](#).

| HWU159 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR1_EL2 , bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR1_EL2 , bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD1 is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD1 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU062, bit [46]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using [TTBR0_EL1](#).

| HWU062 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR0_EL1 , bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR0_EL1 , bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU061, bit [45]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using [TTBR0_EL1](#).

| HWU061 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR0_EL1 , bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR0_EL1 , bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU060, bit [44]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using [TTBR0_EL1](#).

| HWU060 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR0_EL1 , bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR0_EL1 , bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU059, bit [43]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using [TTBR0_EL1](#).

| HWU059 | Meaning |
|--------|---|
| 0b0 | For translations using TTBR0_EL1 , bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR0_EL1 , bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL2.HPD0 is 1. |

The Effective value of this field is 0 if the value of TCR_EL2.HPD0 is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HPD1, bit [42]**When FEAT_HPDS is implemented:**

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by [TTBR1_EL2](#).

| HPD1 | Meaning |
|------|--|
| 0b0 | Hierarchical permissions are enabled. |
| 0b1 | Hierarchical permissions are disabled. |

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HPD0, bit [41]**When FEAT_HPDS is implemented:**

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by [TTBR0_EL2](#).

| HPD0 | Meaning |
|------|--|
| 0b0 | Hierarchical permissions are enabled. |
| 0b1 | Hierarchical permissions are disabled. |

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HD, bit [40]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 1 translations from EL2.

| HD | Meaning |
|-----|--|
| 0b0 | Stage 1 hardware management of dirty state disabled. |
| 0b1 | Stage 1 hardware management of dirty state enabled, only if the HA bit is also set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [39]

When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 1 translations from EL2.

| HA | Meaning |
|-----|--------------------------------------|
| 0b0 | Stage 1 Access flag update disabled. |
| 0b1 | Stage 1 Access flag update enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI1, bit [38]

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the [TTBR1_EL2](#) region, or ignored and used for tagged addresses.

For more information, see 'Address tagging in AArch64 state'.

| TBI1 | Meaning |
|------|--|
| 0b0 | Top Byte used in the address calculation. |
| 0b1 | Top Byte ignored in the address calculation. |

This affects addresses generated in EL0 and EL2 using AArch64 where the address would be translated by tables pointed to by [TTBR1_EL2](#). It has an effect whether the EL2, or EL2&0, translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL2.TBID1 is 1, then this field only applies to Data accesses.

If the value of TBI1 is 1 and bit [55] of the target address to be stored to the PC is 1, then bits[63:56] of that target address are also set to 1 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TBIO, bit [37]

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the [TTBR0_EL2](#) region, or ignored and used for tagged addresses.

For more information, see 'Address tagging in AArch64 state'.

| TBIO | Meaning |
|------|--|
| 0b0 | Top Byte used in the address calculation. |
| 0b1 | Top Byte ignored in the address calculation. |

This affects addresses generated in EL0 and EL2 using AArch64 where the address would be translated by tables pointed to by [TTBR0_EL2](#). It has an effect whether the EL2, or EL2&0, translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL2.TBID0 is 1, then this field only applies to Data accesses.

If the value of TBIO is 1 and bit [55] of the target address to be stored to the PC is 0, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL0 or EL1.
- An exception taken to EL1.
- An exception return to EL0 or EL1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AS, bit [36]

ASID Size.

| AS | Meaning |
|-----|--|
| 0b0 | 8 bit - the upper 8 bits of TTBR0_EL2 and TTBR1_EL2 are ignored by hardware for every purpose except reading back the register, and are treated as if they are all zeros for when used for allocation and matching entries in the TLB. |
| 0b1 | 16 bit - the upper 16 bits of TTBR0_EL2 and TTBR1_EL2 are used for allocation and matching in the TLB. |

If the implementation has only 8 bits of ASID, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [35]

Reserved, RES0.

IPS, bits [34:32]

Intermediate Physical Address Size.

| IPS | Meaning | Applies when |
|-------|-----------------|------------------------------|
| 0b000 | 32 bits, 4GB. | |
| 0b001 | 36 bits, 64GB. | |
| 0b010 | 40 bits, 1TB. | |
| 0b011 | 42 bits, 4TB. | |
| 0b100 | 44 bits, 16TB. | |
| 0b101 | 48 bits, 256TB. | |
| 0b110 | 52 bits, 4PB. | When FEAT_LPA is implemented |

All other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

If the translation granule is not 64KB, the value 0b110 is treated as reserved.

It is IMPLEMENTATION DEFINED whether an implementation that does not implement FEAT_LPA supports setting the value of 0b110 for the 64KB translation granule size or whether setting this value behaves as the 0b101 encoding.

In an implementation that supports 52-bit PAs, if the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by TCR_EL2 are 0b0000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG1, bits [31:30]

Granule size for the [TTBR1_EL2](#).

| TG1 | Meaning |
|------|---------|
| 0b01 | 16KB. |
| 0b10 | 4KB. |
| 0b11 | 64KB. |

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH1, bits [29:28]

Shareability attribute for memory associated with translation table walks using [TTBR1_EL2](#).

| SH1 | Meaning |
|------|------------------|
| 0b00 | Non-shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN1, bits [27:26]

Outer cacheability attribute for memory associated with translation table walks using [TTBR1_EL2](#).

| ORGN1 | Meaning |
|-------|---|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN1, bits [25:24]

Inner cacheability attribute for memory associated with translation table walks using [TTBR1_EL2](#).

| IRGN1 | Meaning |
|-------|---|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EPD1, bit [23]

Translation table walk disable for translations using [TTBR1_EL2](#). This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using [TTBR1_EL2](#). The encoding of this bit is:

| EPD1 | Meaning |
|------|--|
| 0b0 | Perform translation table walks using TTBR1_EL2 . |
| 0b1 | A TLB miss on an address that is translated using TTBR1_EL2 generates a Translation fault. No translation table walk is performed. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A1, bit [22]

Selects whether [TTBR0_EL2](#) or [TTBR1_EL2](#) defines the ASID. The encoding of this bit is:

| A1 | Meaning |
|-----|---|
| 0b0 | TTBR0_EL2 .ASID defines the ASID. |
| 0b1 | TTBR1_EL2 .ASID defines the ASID. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T1SZ, bits [21:16]

The size offset of the memory region addressed by [TTBR1_EL2](#). The region size is $2^{(64-T1SZ)}$ bytes.

The maximum and minimum possible values for T1SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the [TTBR0_EL2](#).

| TG0 | Meaning |
|------------|----------------|
| 0b00 | 4KB. |
| 0b01 | 64KB. |
| 0b10 | 16KB. |

Other values are reserved.

If the value is programmed to either a reserved value, or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using [TTBR0_EL2](#).

| SH0 | Meaning |
|------------|------------------|
| 0b00 | Non-shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using [TTBR0_EL2](#).

| ORGN0 | Meaning |
|--------------|---|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using [TTBR0_EL2](#).

| IRGN0 | Meaning |
|-------|---|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EPD0, bit [7]

Translation table walk disable for translations using [TTBR0_EL2](#). This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using [TTBR0_EL2](#). The encoding of this bit is:

| EPD0 | Meaning |
|------|--|
| 0b0 | Perform translation table walks using TTBR0_EL2 . |
| 0b1 | A TLB miss on an address that is translated using TTBR0_EL2 generates a Translation fault. No translation table walk is performed. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

T0SZ, bits [5:0]

The size offset of the memory region addressed by [TTBR0_EL2](#). The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TCR_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic TCR_EL2 or TCR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0000 | 0b010 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return TCR_EL2;
elsif PSTATE.EL == EL3 then
    return TCR_EL2;

```

MSR TCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    TCR_EL2 = X[t];

```

MRS <Xt>, TCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x120];
    else
        return TCR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TCR_EL2;
    else
        return TCR_EL1;
elsif PSTATE.EL == EL3 then
    return TCR_EL1;

```

MSR TCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.TCR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x120] = X[t];
    else
        TCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        TCR_EL2 = X[t];
    else
        TCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TCR_EL1 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCR_EL3, Translation Control Register (EL3)

The TCR_EL3 characteristics are:

Purpose

The control register for stage 1 of the EL3 translation regime.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to TCR_EL3 are UNDEFINED.

Attributes

TCR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|------|-------|-------|-------|-------|-----|------|----|----|-----|------|----|-----|-----|-------|-------|------|------|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES1 | TCMA | TBID | HWU62 | HWU61 | HWU60 | HWU59 | HPD | RES1 | HD | HA | TBI | RES0 | PS | TG0 | SH0 | ORGNO | IRGN0 | RES0 | TOSZ | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | |

Any of the bits in TCR_EL3 are permitted to be cached in a TLB.

Bits [63:33]

Reserved, RES0.

DS, bit [32]

When FEAT_LPA2 is implemented:

This field affects 52-bit output addressing when using 4KB and 16KB translation granules in stage 1 of the EL3 translation regime.

| DS | Meaning |
|-----|---|
| 0b0 | <p>Bits[49:48] of translation descriptors are RES0.</p> <p>Bits[9:8] in block and page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in table descriptors are ignored by hardware.</p> <p>The minimum value of TCR_EL3.T0SZ is 16. Any memory access using a smaller value generates a stage 1 level 0 translation table fault.</p> <p>Output address[51:48] is 0b0000.</p> |
| 0b1 | <p>Bits[49:48] of translation descriptors hold output address[49:48].</p> <p>Bits[9:8] of table translation descriptors hold output address[51:50].</p> <p>The shareability information of block and page descriptors for cacheable locations is determined by TCR_EL3.SH0.</p> <p>The minimum value of TCR_EL3.T0SZ is 12. Any memory access using a smaller value generates a stage 1 level 0 translation table fault.</p> <p>All calculations of the stage 1 base address are modified for tables of fewer than 8 entries so that the table is aligned to 64 bytes.</p> <p>Bits[5:2] of TTBR0_EL3 are used to hold bits[51:48] of the output address in all cases.</p> <hr/> <p>Note</p> <p>As FEAT_LVA must be implemented if TCR_EL3.DS == 1, the minimum value of the TCR_EL3.T0SZ field is 12, as determined by that extension.</p> <hr/> <p>For the TLBI Range instructions affecting VA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0b0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 0b00.</p> <hr/> <p>Note</p> <p>This forces alignment of the ranges used by the TLBI range instructions.</p> <hr/> |

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [31]

Reserved, RES1.

TCMA, bit [30]

When FEAT_MTE2 is implemented:

Controls the generation of Unchecked accesses at EL3 when address [59:56] = 0b0000.

| TCMA | Meaning |
|------|---|
| 0b0 | This control has no effect on the generation of Unchecked accesses. |
| 0b1 | All accesses are Unchecked. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBID, bit [29]

When FEAT_PAAuth is implemented:

Controls the use of the top byte of instruction addresses for address matching.

| TBID | Meaning |
|------|---|
| 0b0 | TCR_EL3.TBI applies to Instruction and Data accesses. |
| 0b1 | TCR_EL3.TBI applies to Data accesses only. |

This affects addresses where the address would be translated by tables pointed to by [TTBR0_EL3](#).

For the purpose of this field, all cache maintenance and address translation instructions that perform address translation are treated as data accesses.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry.

| HWU62 | Meaning |
|-------|---|
| 0b0 | Bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1. |

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU61, bit [27]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry.

| HWU61 | Meaning |
|-------|---|
| 0b0 | Bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1. |

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU60, bit [26]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry.

| HWU60 | Meaning |
|-------|---|
| 0b0 | Bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1. |

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU59, bit [25]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry.

| HWU59 | Meaning |
|-------|---|
| 0b0 | Bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TCR_EL3.HPD is 1. |

The Effective value of this field is 0 if the value of TCR_EL3.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HPD, bit [24]**When FEAT_HPDS is implemented:**

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, PXNTable, and UXNTable, except NSTable, in the translation tables pointed to by [TTBR0_EL3](#).

| HPD | Meaning |
|-----|--|
| 0b0 | Hierarchical permissions are enabled. |
| 0b1 | Hierarchical permissions are disabled. |

Note
In this case, bit[61] (APTable[0]) and bit[59] (PXNTable) of the next level descriptor attributes are required to be ignored by the PE, and are no longer reserved, allowing them to be used by software.

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES1.

HD, bit [22]**When FEAT_HAFDBS is implemented:**

Hardware management of dirty state in stage 1 translations from EL3.

| HD | Meaning |
|-----|--|
| 0b0 | Stage 1 hardware management of dirty state disabled. |
| 0b1 | Stage 1 hardware management of dirty state enabled, only if the HA bit is also set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]**When FEAT_HAFDBS is implemented:**

Hardware Access flag update in stage 1 translations from EL3.

| HA | Meaning |
|-----|--------------------------------------|
| 0b0 | Stage 1 Access flag update disabled. |
| 0b1 | Stage 1 Access flag update enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TBI, bit [20]

Top Byte Ignored. Indicates whether the top byte of an address is used for address match for the [TTBR0_EL3](#) region, or ignored and used for tagged addresses.

| TBI | Meaning |
|-----|--|
| 0b0 | Top Byte used in the address calculation. |
| 0b1 | Top Byte ignored in the address calculation. |

This affects addresses generated in EL3 using AArch64 where the address would be translated by tables pointed to by [TTBR0_EL3](#). It has an effect whether the EL3 translation regime is enabled or not.

If FEAT_PAuth is implemented and TCR_EL3.TBID is 1, then this field only applies to Data accesses.

Otherwise, if the value of TBI is 1, then bits[63:56] of that target address are also set to 0 before the address is stored in the PC, in the following cases:

- A branch or procedure return within EL3.
- A exception taken to EL3.
- An exception return to EL3.

For more information, see 'Address tagging in AArch64 state'.

Note

This control determines the scope of address tagging. It never causes an exception to be generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

PS, bits [18:16]

Physical Address Size.

| PS | Meaning |
|-------|-----------------|
| 0b000 | 32 bits, 4GB. |
| 0b001 | 36 bits, 64GB. |
| 0b010 | 40 bits, 1TB. |
| 0b011 | 42 bits, 4TB. |
| 0b100 | 44 bits, 16TB. |
| 0b101 | 48 bits, 256TB. |
| 0b110 | 52 bits, 4PB. |

All other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

If the translation granule is not 64KB and FEAT_LPA2 is not implemented, the value 0b110 is treated as reserved.

It is IMPLEMENTATION DEFINED whether an implementation that does not implement FEAT_LPA supports setting the value of 0b110 for the 64KB translation granule size or whether setting this value behaves as the 0b101 encoding.

In an implementation that supports 52-bit PAs, if the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by TCR_EL3 are 0b0000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the [TTBR0_EL3](#).

| TG0 | Meaning |
|------|---------|
| 0b00 | 4KB. |
| 0b01 | 64KB. |
| 0b10 | 16KB. |

Other values are reserved.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using [TTBR0_EL3](#).

| SH0 | Meaning |
|------|------------------|
| 0b00 | Non-shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using [TTBR0_EL3](#).

| ORGN0 | Meaning |
|-------|---|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using [TTBR0_EL3](#).

| IRGN0 | Meaning |
|-------|---|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:6]

Reserved, RES0.

TOSZ, bits [5:0]

The size offset of the memory region addressed by [TTBR0_EL3](#). The region size is $2^{(64-TOSZ)}$ bytes.

The maximum and minimum possible values for TOSZ depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TCR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0010 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return TCR_EL3;

```

MSR TCR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0010 | 0b0000 | 0b010 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TCR_EL3 = X[t];
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TFSR_EL1, Tag Fault Status Register (EL1)

The TFSR_EL1 characteristics are:

Purpose

Holds accumulated Tag Check Faults occurring in EL1 that are not taken precisely.

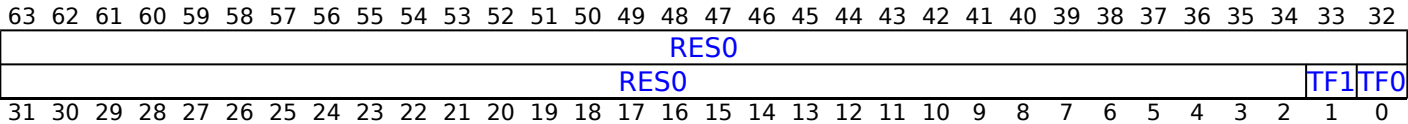
Configuration

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to TFSR_EL1 are UNDEFINED.

Attributes

TFSR_EL1 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

TF1, bit [1]

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == 0b1 occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TF0, bit [0]

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == 0b0 occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TFSR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TFSR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x190];
    else
        return TFSR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return TFSR_EL2;
    else
        return TFSR_EL1;
elsif PSTATE.EL == EL3 then
    return TFSR_EL1;

```

MSR TFSR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x190] = X[t];
    else
        TFSR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        TFSR_EL2 = X[t];
    else
        TFSR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TFSR_EL1 = X[t];

```

MRS <Xt>, TFSR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0101 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x190];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TFSR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return TFSR_EL1;
    else
        UNDEFINED;

```

MSR TFSR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0101 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x190] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        TFSR_EL1 = X[t];
    else
        UNDEFINED;

```

MRS <Xt>, TFSR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif EL2Enabled() && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TFSR_EL1;
        elsif EL2Enabled() && HCR_EL2.NV == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TFSR_EL2;
    elsif PSTATE.EL == EL3 then
        return TFSR_EL2;

```

MSR TFSR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif EL2Enabled() && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                TFSR_EL1 = X[t];
        elsif EL2Enabled() && HCR_EL2.NV == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        TFSR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TFSR_EL2, Tag Fault Status Register (EL2)

The TFSR_EL2 characteristics are:

Purpose

Holds accumulated Tag Check Faults occurring in EL2 that are not taken precisely.

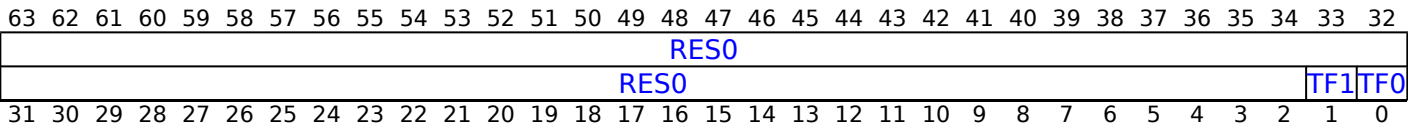
Configuration

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to TFSR_EL2 are UNDEFINED.

Attributes

TFSR_EL2 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

TF1, bit [1]

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == 0b1 occurs.

When [HCR_EL2.E2H](#)==0b0, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TF0, bit [0]

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == 0b0 occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TFSR_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic TFSR_EL2 or TFSR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TFSR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif EL2Enabled() && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TFSR_EL1;
        elsif EL2Enabled() && HCR_EL2.NV == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TFSR_EL2;
    elsif PSTATE.EL == EL3 then
        return TFSR_EL2;

```

MSR TFSR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif EL2Enabled() && HCR_EL2.ATA == '0' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                TFSR_EL1 = X[t];
        elsif EL2Enabled() && HCR_EL2.NV == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            UNDEFINED;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSR_EL2 = X[t];
    elsif PSTATE.EL == EL3 then
        TFSR_EL2 = X[t];

```

MRS <Xt>, TFSR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x190];
    else
        return TFSR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return TFSR_EL2;
    else
        return TFSR_EL1;
elsif PSTATE.EL == EL3 then
    return TFSR_EL1;

```

MSR TFSR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x190] = X[t];
    else
        TFSR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        TFSR_EL2 = X[t];
    else
        TFSR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TFSR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TFSR_EL3, Tag Fault Status Register (EL3)

The TFSR_EL3 characteristics are:

Purpose

Holds accumulated Tag Check Faults occurring in EL3 that are not taken precisely.

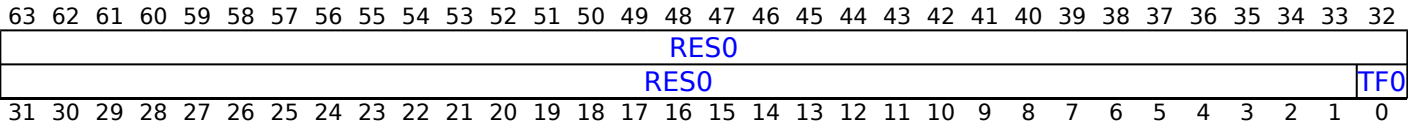
Configuration

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to TFSR_EL3 are UNDEFINED.

Attributes

TFSR_EL3 is a 64-bit register.

Field descriptions



Bits [63:1]

Reserved, RES0.

TF0, bit [0]

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == 0b0 occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TFSR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TFSR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0101 | 0b0110 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return TFSR_EL3;
```

MSR TFSR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0101 | 0b0110 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TFSR_EL3 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TFSRE0_EL1, Tag Fault Status Register (EL0).

The TFSRE0_EL1 characteristics are:

Purpose

Holds accumulated Tag Check Faults occurring in EL0 that are not taken precisely.

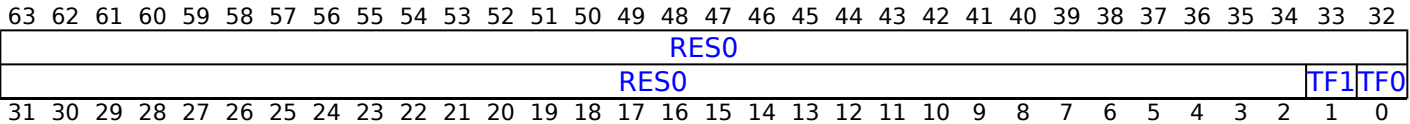
Configuration

This register is present only when FEAT_MTE2 is implemented. Otherwise, direct accesses to TFSRE0_EL1 are UNDEFINED.

Attributes

TFSRE0_EL1 is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

TF1, bit [1]

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == 0b1 occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TF0, bit [0]

Tag Check Fault. Asynchronously set to 1 when a Tag Check Fault using a virtual address with bit[55] == 0b0 occurs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TFSRE0_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TFSRE0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TFSRE0_EL1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TFSRE0_EL1;
    elsif PSTATE.EL == EL3 then
        return TFSRE0_EL1;

```

MSR TFSRE0_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0101 | 0b0110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.ATA == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSRE0_EL1 = X[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.ATA == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && SCR_EL3.ATA == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TFSRE0_EL1 = X[t];
    elsif PSTATE.EL == EL3 then
        TFSRE0_EL1 = X[t];

```


TLBI ALLE1, TLBI ALLE1NXS, TLB Invalidate All, EL1

The TLBI ALLE1, TLBI ALLE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate an address using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate an address using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to entries with any VMID.

The invalidation only applies to the PE that executes this System instruction.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI ALLE1, TLBI ALLE1NXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI ALLE1, TLBI ALLE1NXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI ALLE1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0111 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_NSH, TLBI_AllAttr);
elseif PSTATE.EL == EL3 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_NSH, TLBI_AllAttr);

```

TLBI ALLE1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0111 | 0b100 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elseif PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_NSH, TLBI_ExcludeXS);
elseif PSTATE.EL == EL3 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_NSH, TLBI_ExcludeXS);

```

TLBI ALLE1IS, TLBI ALLE1ISNXS, TLB Invalidate All, EL1, Inner Shareable

The TLBI ALLE1IS, TLBI ALLE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate an address using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate an address using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to entries with any VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI ALLE1IS, TLBI ALLE1ISNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI ALLE1IS, TLBI ALLE1ISNXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI ALLE1IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0011 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_ISH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_ISH, TLBI_AllAttr);

```

TLBI ALLE1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0011 | 0b100 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_ISH, TLBI_ExcludeXS);
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_ISH, TLBI_ExcludeXS);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ALLE1OS, TLBI ALLE1OSNXS, TLB Invalidate All, EL1, Outer Shareable

The TLBI ALLE1OS, TLBI ALLE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate an address using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate an address using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to entries with any VMID.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI ALLE1OS, TLBI ALLE1OSNXS are UNDEFINED.

Attributes

TLBI ALLE1OS, TLBI ALLE1OSNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI ALLE1OS, TLBI ALLE1OSNXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI ALLE1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0001 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_OSH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_OSH, TLBI_AllAttr);

```

TLBI ALLE1OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0001 | 0b100 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_OSH, TLBI_ExcludeXS);
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_OSH, TLBI_ExcludeXS);

```

TLBI ALLE2, TLBI ALLE2NXS, TLB Invalidate All, EL2

The TLBI ALLE2, TLBI ALLE2NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate an address using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate an address using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI ALLE2, TLBI ALLE2NXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI ALLE2, TLBI ALLE2NXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONstrained UNpredictable whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI ALLE2{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0111 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Shareability_NSH, TLBI_AllAttr);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_NSH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Shareability_NSH, TLBI_AllAttr);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_NSH, TLBI_AllAttr);

```

TLBI ALLE2NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0111 | 0b000 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Shareability_NSH, TLBI_ExcludeXS);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_NSH, TLBI_ExcludeXS);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Shareability_NSH, TLBI_ExcludeXS);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_NSH, TLBI_ExcludeXS);

```

TLBI ALLE2IS, TLBI ALLE2ISNXS, TLB Invalidate All, EL2, Inner Shareable

The TLBI ALLE2IS, TLBI ALLE2ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate an address using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate an address using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI ALLE2IS, TLBI ALLE2ISNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI ALLE2IS, TLBI ALLE2ISNXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI ALLE2IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Shareability_ISH, TLBI_AllAttr);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_ISH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Shareability_ISH, TLBI_AllAttr);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_ISH, TLBI_AllAttr);

```

TLBI ALLE2ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0011 | 0b000 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Shareability_ISH, TLBI_ExcludeXS);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_ISH, TLBI_ExcludeXS);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Shareability_ISH, TLBI_ExcludeXS);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_ISH, TLBI_ExcludeXS);

```

TLBI ALLE2OS, TLBI ALLE2OSNXS, TLB Invalidate All, EL2, Outer Shareable

The TLBI ALLE2OS, TLBI ALLE2OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate an address using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate an address using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate an address using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate an address using the Non-secure EL1&0 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI ALLE2OS, TLBI ALLE2OSNXS are UNDEFINED.

Attributes

TLBI ALLE2OS, TLBI ALLE2OSNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI ALLE2OS, TLBI ALLE2OSNXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.

- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI ALLE2OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Shareability_OSH, TLBI_AllAttr);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_OSH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Shareability_OSH, TLBI_AllAttr);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_OSH, TLBI_AllAttr);

```

TLBI ALLE2OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0001 | 0b000 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Shareability_OSH, TLBI_ExcludeXS);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_OSH, TLBI_ExcludeXS);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL20, Shareability_OSH, TLBI_ExcludeXS);
    else
        AArch64.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_OSH, TLBI_ExcludeXS);

```

TLBI ALLE3, TLBI ALLE3NXS, TLB Invalidate All, EL3

The TLBI ALLE3, TLBI ALLE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be required to translate an address using the EL3 translation regime.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI ALLE3, TLBI ALLE3NXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI ALLE3, TLBI ALLE3NXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI ALLE3{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1000 | 0b0111 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL3, Shareability_NSH, TLBI_AllAttr);

```

TLBI ALLE3NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0111 | 0b000 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL3, Shareability_NSH, TLBI_ExcludeXS);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ALLE3IS, TLBI ALLE3ISNXS, TLB Invalidate All, EL3, Inner Shareable

The TLBI ALLE3IS, TLBI ALLE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be required to translate an address using the EL3 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI ALLE3IS, TLBI ALLE3ISNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI ALLE3IS, TLBI ALLE3ISNXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI ALLE3IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1000 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL3, Shareability_ISH, TLBI_AllAttr);

```

TLBI ALLE3ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0011 | 0b000 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL3, Shareability_ISH, TLBI_ExcludeXS);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ALLE3OS, TLBI ALLE3OSNXS, TLB Invalidate All, EL3, Outer Shareable

The TLBI ALLE3OS, TLBI ALLE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be required to translate an address using the EL3 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI ALLE3OS, TLBI ALLE3OSNXS are UNDEFINED.

Attributes

TLBI ALLE3OS, TLBI ALLE3OSNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI ALLE3OS, TLBI ALLE3OSNXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI ALLE3OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1000 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL3, Shareability_OSH, TLBI_AllAttr);

```

TLBI ALLE3OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0001 | 0b000 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL3, Shareability_OSH, TLBI_ExcludeXS);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ASIDE1, TLBI ASIDE1NXS, TLB Invalidate by ASID, EL1

The TLBI ASIDE1, TLBI ASIDE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
 - Is from a level of lookup above the final level.
 - Is a non-global entry from the final level of lookup.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate an address using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate an address using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate an address using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI ASIDE1, TLBI ASIDE1NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

ASID value to match. Any appropriate TLB entries that match the ASID values will be affected by this System instruction.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Bits [47:0]

Reserved, RES0.

Executing the TLBI ASIDE1, TLBI ASIDE1NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI ASIDE1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0111 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIASIDE1 == '1'
    then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
        then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBI_AllAttr, X[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
            && HCRX_EL2.FnXS == '1' then
                AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL2 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                    TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                    TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL3 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                    TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                    TLBI_AllAttr, X[t]);

```

TLBI ASIDE1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0111 | 0b010 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIASIDE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
            TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ASIDE1IS, TLBI ASIDE1ISNXS, TLB Invalidate by ASID, EL1, Inner Shareable

The TLBI ASIDE1IS, TLBI ASIDE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
 - Is from a level of lookup above the final level.
 - Is a non-global entry from the final level of lookup.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate an address using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate an address using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate an address using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI ASIDE1IS, TLBI ASIDE1ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

ASID value to match. Any appropriate TLB entries that match the ASID values will be affected by this System instruction.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Bits [47:0]

Reserved, RES0.

Executing the TLBI ASIDE1IS, TLBI ASIDE1ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI ASIDE1IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIASIDE1IS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_AllAttr, X[t]);

```

TLBI ASIDE1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0011 | 0b010 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIASIDE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
            TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
                TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
                TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI ASIDE1OS, TLBI ASIDE1OSNXS, TLB Invalidate by ASID, EL1, Outer Shareable

The TLBI ASIDE1OS, TLBI ASIDE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
 - Is from a level of lookup above the final level.
 - Is a non-global entry from the final level of lookup.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate an address using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate an address using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate an address using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI ASIDE1OS, TLBI ASIDE1OSNXS are UNDEFINED.

Attributes

TLBI ASIDE1OS, TLBI ASIDE1OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

ASID value to match. Any appropriate TLB entries that match the ASID values will be affected by this System instruction.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Bits [47:0]

Reserved, RES0.

Executing the TLBI ASIDE1OS, TLBI ASIDE1OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI ASIDE1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIASIDE1OS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBI_AllAttr, X[t]);

```

TLBI ASIDE1OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0001 | 0b010 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
&& (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIASIDE10S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_ASID(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI IPAS2E1, TLBI IPAS2E1NXS, TLB Invalidate by Intermediate Physical Address, Stage 2, EL1

The TLBI IPAS2E1, TLBI IPAS2E1NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidation of TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI IPAS2E1, TLBI IPAS2E1NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|------|----|----|------------|----|----|----|------------|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| NS | | RES0 | | | | | | | | | | | | | | | TTL | | | RES0 | | | IPA[51:48] | | | | IPA[47:12] | | | | |
| IPA[47:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and [SCR_EL3](#).{NSE, NS} == {0, 0}, NS selects the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed and [SCR_EL3](#).{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and [SCR_EL3](#).{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|------------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:40]

Reserved, RES0.

IPA[51:48], bits [39:36]

When FEAT_LPA is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When FEAT_LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2E1, TLBI IPAS2E1NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI IPAS2E1{, <Xt>}

| | | | | |
|------------|------------|------------|------------|------------|
| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|

| | | | | |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b001 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBILevel_Any, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI IPAS2E1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBILevel_Any, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBILevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI IPAS2E1IS, TLBI IPAS2E1ISNXS, TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

The TLBI IPAS2E1IS, TLBI IPAS2E1ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidation of TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

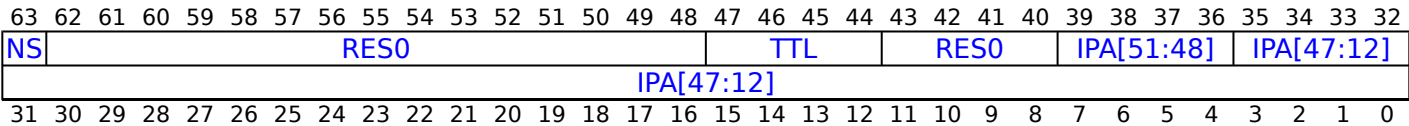
Configuration

There are no configuration notes.

Attributes

TLBI IPAS2E1IS, TLBI IPAS2E1ISNXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:40]

Reserved, RES0.

IPA[51:48], bits [39:36]
When FEAT_LPA is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When FEAT_LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2E1IS, TLBI IPAS2E1ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI IPAS2E1IS{, <Xt>}

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0000 | 0b001 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Any, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Any, TLBI_AllAttr, X[t]);

```

TLBI IPAS2E1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0000 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI IPAS2E1OS, TLBI IPAS2E1OSNXS, TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

The TLBI IPAS2E1OS, TLBI IPAS2E1OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidation of TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

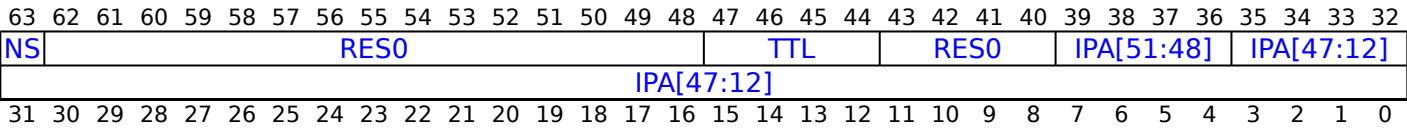
Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI IPAS2E1OS, TLBI IPAS2E1OSNXS are UNDEFINED.

Attributes

TLBI IPAS2E1OS, TLBI IPAS2E1OSNXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]
When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]
When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:40]

Reserved, RES0.

IPA[51:48], bits [39:36]

Extension to IPA[47:12]. For more information, see IPA[47:12].

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When FEAT_LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2E1OS, TLBI IPAS2E1OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI IPAS2E1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBILevel_Any, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI IPAS2E1OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b000 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBILevel_Any, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBILevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI IPAS2LE1, TLBI IPAS2LE1NXS, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

The TLBI IPAS2LE1, TLBI IPAS2LE1NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidation of TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI IPAS2LE1, TLBI IPAS2LE1NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|------|----|----|----|------------|----|----|----|------------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| NS | | RES0 | | | | | | | | | | | | | | TTL | | | | RES0 | | | | IPA[51:48] | | | | IPA[47:12] | | | |
| IPA[47:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:40]

Reserved, RES0.

IPA[51:48], bits [39:36]
When FEAT_LPA is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When FEAT_LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2LE1, TLBI IPAS2LE1NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI IPAS2LE1{, <Xt>}

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b101 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBIlevel_Last, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBIlevel_Last, TLBI_AllAttr, X[t]);

```

TLBI IPAS2LE1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI IPAS2LE1IS, TLBI IPAS2LE1ISNXS, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

The TLBI IPAS2LE1IS, TLBI IPAS2LE1ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidation of TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

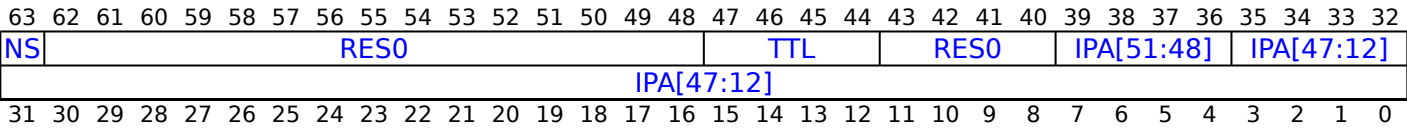
Configuration

There are no configuration notes.

Attributes

TLBI IPAS2LE1IS, TLBI IPAS2LE1ISNXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]
When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]
When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:40]

Reserved, RES0.

IPA[51:48], bits [39:36]
When FEAT_LPA is implemented:

Extension to IPA[47:12]. For more information, see IPA[47:12].

Otherwise:

Reserved, RES0.

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When FEAT_LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2LE1IS, TLBI IPAS2LE1ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI IPAS2LE1IS{, <Xt>}
```

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0000 | 0b101 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Last, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Last, TLBI_AllAttr, X[t]);

```

TLBI IPAS2LE1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0000 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI IPAS2LE1OS, TLBI IPAS2LE1OSNXS, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

The TLBI IPAS2LE1OS, TLBI IPAS2LE1OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

For more information about the architectural requirements for this System instruction, see 'Invalidation of TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

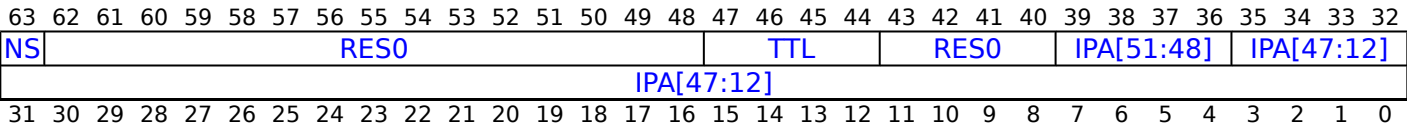
Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI IPAS2LE1OS, TLBI IPAS2LE1OSNXS are UNDEFINED.

Attributes

TLBI IPAS2LE1OS, TLBI IPAS2LE1OSNXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]
When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TTL, bits [47:44]
When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

Bits [43:40]

Reserved, RES0.

IPA[51:48], bits [39:36]

Extension to IPA[47:12]. For more information, see IPA[47:12].

IPA[47:12], bits [35:0]

Bits[47:12] of the intermediate physical address to match. For implementations with fewer than 48 bits, the upper bits of this field are RES0.

When FEAT_LPA is implemented, and 52-bit addresses and a 64KB translation granule are in use, IPA[51:48] form the upper part of the address value. Otherwise, IPA[51:48] are RES0.

Executing the TLBI IPAS2LE1OS, TLBI IPAS2LE1OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI IPAS2LE1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBIlevel_Last, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBIlevel_Last, TLBI_AllAttr, X[t]);

```

TLBI IPAS2LE1OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b100 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI PAALL, TLB Invalidate GPT Information by PA, All Entries, Local

The TLBI PAALL characteristics are:

Purpose

Invalidates cached copies of GPT entries from TLBs. Details:

- The invalidation applies to TLB entries containing GPT information that relates to a physical address.
- The invalidation applies to all TLB entries containing GPT information.
- The invalidation affects only the TLBs for the PE executing the operation.

The full set of TLB maintenance instructions that invalidate cached GPT entries is: [TLBI PAALL](#), [TLBI PAALLOS](#), [TLBI RPALOS](#), and [TLBI RPAOS](#).

These instructions have the same ordering, observability, and completion behavior as all other TLBI instructions.

Configuration

This instruction is present only when FEAT_RME is implemented. Otherwise, direct accesses to TLBI PAALL are UNDEFINED.

Attributes

TLBI PAALL is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI PAALL instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI PAALL{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1000 | 0b0111 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_PAALL(Shareability_NSH);

```


TLBI PAALLOS, TLB Invalidate GPT Information by PA, All Entries, Outer Shareable

The TLBI PAALLOS characteristics are:

Purpose

Invalidates cached copies of GPT entries from TLBs. Details:

- The invalidation applies to TLB entries containing GPT information that relates to a physical address.
- The invalidation applies to all TLB entries containing GPT information.
- The invalidation affects all TLBs in the Outer Shareable domain.

The full set of TLB maintenance instructions that invalidate cached GPT entries is: [TLBI PAALL](#), [TLBI PAALLOS](#), [TLBI RPALOS](#), and [TLBI RPAOS](#).

These instructions have the same ordering, observability, and completion behavior as all other TLBI instructions.

Configuration

This instruction is present only when FEAT_RME is implemented. Otherwise, direct accesses to TLBI PAALLOS are UNDEFINED.

Attributes

TLBI PAALLOS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI PAALLOS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI PAALLOS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1000 | 0b0001 | 0b100 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_PAALL(Shareability_OSH);
```


TLBI RIPAS2E1, TLBI RIPAS2E1NXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1

The TLBI RIPAS2E1, TLBI RIPAS2E1NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 0000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidation of TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

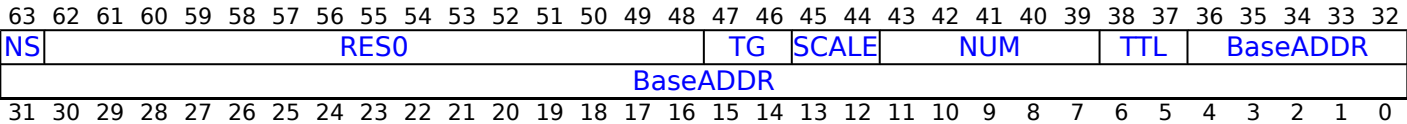
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RIPAS2E1, TLBI RIPAS2E1NXS are UNDEFINED.

Attributes

TLBI RIPAS2E1, TLBI RIPAS2E1NXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]

When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|--|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RIPAS2E1, TLBI RIPAS2E1NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RIPAS2E1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBILevel_Any, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI RIPAS2E1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b010 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBILevel_Any, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBILevel_Any, TLBI_ExcludeXS, X[t]);

```

TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

The TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidation of TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS are UNDEFINED.

Attributes

TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----------|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| NS | RES0 | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | BaseADDR | | | | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

NS, bit [63]
When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|--|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RIPAS2E1IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBILevel_Any, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI RIPAS2E1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0000 | 0b010 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBILevel_Any, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBILevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable

The TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidation of TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS are UNDEFINED.

Attributes

TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----------|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| NS | RES0 | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | BaseADDR | | | | | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

NS, bit [63]
When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|--|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RIPAS2E1OS{, <Xt>}
```

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBILevel_Any, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI RIPAS2E1OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b011 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBILevel_Any, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBILevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

The TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula $[BaseAddr \leq VA < BaseAddr + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation only applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 0000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidation of TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

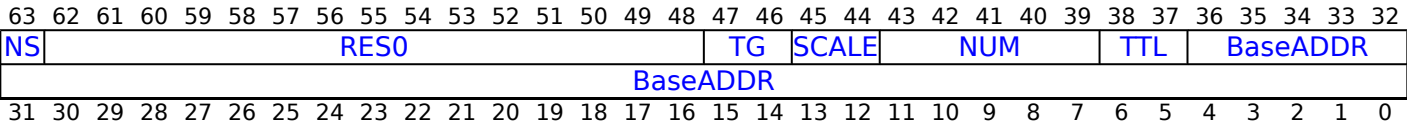
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS are UNDEFINED.

Attributes

TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]
When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RIPAS2LE1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBIlevel_Last, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBIlevel_Last, TLBI_AllAttr, X[t]);

```

TLBI RIPAS2LE1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b110 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable

The TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 00000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidation of TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

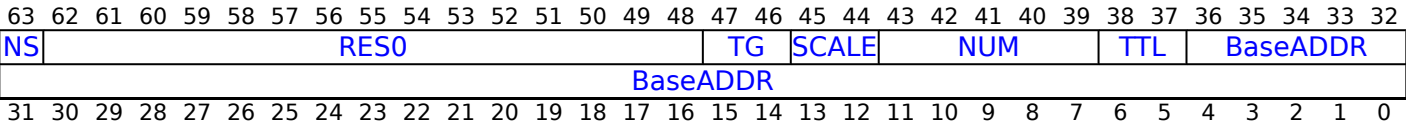
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS are UNDEFINED.

Attributes

TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS is a 64-bit System instruction.

Field descriptions



NS, bit [63]
When FEAT_RME is implemented:

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 0}, NS selects the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed and SCR_EL3.{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and SCR_EL3.{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

```
TLBI RIPAS2LE1IS{, <Xt>}
```

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Last, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Last, TLBI_AllAttr, X[t]);

```

TLBI RIPAS2LE1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0000 | 0b110 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

The TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS characteristics are:

Purpose

If EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - [SCR_EL3](#).{NSE, NS} is {0, 0} and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {0, 1} and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
 - [SCR_EL3](#).{NSE, NS} is {1, 1} and the entry would be required to translate the specified IPA using the Realm EL1&0 translation regime.
- If FEAT_RME is not implemented, one of the following applies:
 - [SCR_EL3](#).NS is 0 and the entry would be required to translate the specified IPA using the Secure EL1&0 translation regime.
 - [SCR_EL3](#).NS is 1 and the entry would be required to translate the specified IPA using the Non-secure EL1&0 translation regime.
- The entry would be used with the current VMID.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:

TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS, TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable

- If $TTL == 10$ and $BaseADDR[24:14]$ is not equal to 00000000000.
- For the 64K translation granule:
 - If $TTL == 01$ and $BaseADDR[41:16]$ is not equal to 00000000000000000000000000000000.
 - If $TTL == 10$ and $BaseADDR[28:16]$ is not equal to 0000000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidation of TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS are UNDEFINED.

Attributes

TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|----|-----|----------|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| NS | RES0 | | | | | | | | | | | | | | TG | SCALE | NUM | | | | | TTL | BaseADDR | | | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

NS, bit [63]
When FEAT_RME is implemented:

When the instruction is executed and [SCR_EL3](#).{NSE, NS} == {0, 0}, NS selects the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed and [SCR_EL3](#).{NSE, NS} == {1, 1}, this field is RES0, and the instruction applies only to the Realm IPA space.

When the instruction is executed and [SCR_EL3](#).{NSE, NS} == {0, 1}, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is implemented and FEAT_RME is not implemented:

Not Secure. Specifies the IPA space.

| NS | Meaning |
|-----|-------------------------------------|
| 0b0 | IPA is in the Secure IPA space. |
| 0b1 | IPA is in the Non-secure IPA space. |

When the instruction is executed in Non-secure state, this field is RES0, and the instruction applies only to the Non-secure IPA space.

When FEAT_SEL2 is not implemented, or if EL2 is disabled in the current Security state, this field is RES0.

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RIPAS2LE1OS, TLBI RIPAS2LE1OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RIPAS2LE1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBIlevel_Last, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBIlevel_Last, TLBI_AllAttr, X[t]);

```

TLBI RIPAS2LE1OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b111 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        //no operation
    else
        AArch64.TLBI_RIPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

TLBI RPALOS, TLB Range Invalidate GPT Information by PA, Last level, Outer Shareable

The TLBI RPALOS characteristics are:

Purpose

Invalidates cached copies of GPT entries from TLBs. Details:

- The invalidation applies to TLB entries containing GPT information that relates to a physical address.
- The invalidation affects all TLBs in the Outer Shareable domain.
- Invalidates TLB entries containing GPT information from the final level of the GPT walk that relates to the supplied physical address.
- Invalidations are range-based, invalidating TLB entries starting from the address in BaseADDR, within the range as specified by SIZE.

The full set of TLB maintenance instructions that invalidate cached GPT entries is: [TLBI PAALL](#), [TLBI PAALLOS](#), [TLBI RPALOS](#), and [TLBI RPAOS](#).

These instructions have the same ordering, observability, and completion behavior as all other TLBI instructions.

Configuration

This instruction is present only when FEAT_RME is implemented. Otherwise, direct accesses to TLBI RPALOS are UNDEFINED.

Attributes

TLBI RPALOS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|----|----|---------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | SIZE | | | | RES0 | | | | Address | | | | | | | |
| Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

SIZE, bits [47:44]

Size of the range for invalidation.

If SIZE is a reserved value, no TLB entries are required to be invalidated.

| SIZE | Meaning |
|--------|---------|
| 0b0000 | 4KB. |
| 0b0001 | 16KB. |
| 0b0010 | 64KB. |
| 0b0011 | 2MB. |
| 0b0100 | 32MB. |
| 0b0101 | 512MB. |
| 0b0110 | 1GB. |
| 0b0111 | 16GB. |
| 0b1000 | 64GB. |
| 0b1001 | 512GB. |

All other values are reserved.

If SIZE gives a range smaller than the configured physical granule size in [GPCCR_EL3.PGS](#), then the effective value of SIZE is taken to be the size configured by [GPCCR_EL3.PGS](#).

If [GPCCR_EL3.PGS](#) is configured to a reserved value, no TLB entries are required to be invalidated.

Bits [43:40]

Reserved, RES0.

Address, bits [39:0]

The starting address for the range of the maintenance instruction.

This field is decoded with reference to the value of [GPCCR_EL3.PGS](#) to give BaseADDR as follows:

| GPCCR_EL3.PGS | BaseADDR |
|---------------|----------------------------|
| 0b00 (4KB) | BaseADDR[51:12] = Xt[39:0] |
| 0b10 (16KB) | BaseADDR[51:14] = Xt[39:2] |
| 0b01 (64KB) | BaseADDR[51:16] = Xt[39:4] |

Other bits of BaseADDR are treated as zero, to give the effective value of BaseADDR.

If the effective value of BaseADDR is not aligned to the size of the effective value of SIZE, no TLB entries are required to be invalidated.

If [GPCCR_EL3.PGS](#) is configured to a reserved value, no TLB entries are required to be invalidated.

Executing the TLBI RPALOS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RPALOS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1000 | 0b0100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RPA(TLBIlevel_Last, X[t], Shareability_0SH);

```


TLBI RPAOS, TLB Range Invalidate GPT Information by PA, Outer Shareable

The TLBI RPAOS characteristics are:

Purpose

Invalidates cached copies of GPT entries from TLBs. Details:

- The invalidation applies to TLB entries containing GPT information that relates to a physical address.
- The invalidation affects all TLBs in the Outer Shareable domain.
- Invalidates TLB entries containing GPT information from all levels of the GPT walk that relates to the supplied physical address.
- Invalidations are range-based, invalidating TLB entries starting from the address in BaseADDR, within the range as specified by SIZE.

The full set of TLB maintenance instructions that invalidate cached GPT entries is: [TLBI PAALL](#), [TLBI PAALLOS](#), [TLBI RPALOS](#), and [TLBI RPAOS](#).

These instructions have the same ordering, observability, and completion behavior as all other TLBI instructions.

Configuration

This instruction is present only when FEAT_RME is implemented. Otherwise, direct accesses to TLBI RPAOS are UNDEFINED.

Attributes

TLBI RPAOS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|----|----|---------|----|----|----|----|----|----|----|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| RES0 | | | | | | | | | | | | | | | | SIZE | | | | RES0 | | | | Address | | | | | | | | | | | |
| Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

Bits [63:48]

Reserved, RES0.

SIZE, bits [47:44]

Size of the range for invalidation.

If SIZE is a reserved value, no TLB entries are required to be invalidated.

| SIZE | Meaning |
|--------|---------|
| 0b0000 | 4KB. |
| 0b0001 | 16KB. |
| 0b0010 | 64KB. |
| 0b0011 | 2MB. |
| 0b0100 | 32MB. |
| 0b0101 | 512MB. |
| 0b0110 | 1GB. |
| 0b0111 | 16GB. |
| 0b1000 | 64GB. |
| 0b1001 | 512GB. |

All other values are reserved.

If SIZE gives a range smaller than the configured physical granule size in [GPCCR_EL3.PGS](#), then the effective value of SIZE is taken to be the size configured by [GPCCR_EL3.PGS](#).

If [GPCCR_EL3.PGS](#) is configured to a reserved value, no TLB entries are required to be invalidated.

Bits [43:40]

Reserved, RES0.

Address, bits [39:0]

The starting address for the range of the maintenance instruction.

This field is decoded with reference to the value of [GPCCR_EL3.PGS](#) to give BaseADDR as follows:

| GPCCR_EL3.PGS | BaseADDR |
|---------------|----------------------------|
| 0b00 (4KB) | BaseADDR[51:12] = Xt[39:0] |
| 0b10 (16KB) | BaseADDR[51:14] = Xt[39:2] |
| 0b01 (64KB) | BaseADDR[51:16] = Xt[39:4] |

Other bits of BaseADDR are treated as zero, to give the effective value of BaseADDR.

If the effective value of BaseADDR is not aligned to the size of the effective value of SIZE, no TLB entries are required to be invalidated.

If [GPCCR_EL3.PGS](#) is configured to a reserved value, no TLB entries are required to be invalidated.

Executing the TLBI RPAOS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RPAOS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1000 | 0b0100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RPA(TLBILevel_Any, X[t], Shareability_0SH);

```


TLBI RVAAE1, TLBI RVAAE1NXS, TLB Range Invalidate by VA, All ASID, EL1

The TLBI RVAAE1, TLBI RVAAE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVAAE1, TLBI RVAAE1NXS are UNDEFINED.

Attributes

TLBI RVAAE1, TLBI RVAAE1NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|----|-----|----|----------|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAAE1, TLBI RVAAE1NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAAE1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0110 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVAE1 == '1'
    then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
        then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Any, TLBI_AllAttr, X[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
            && HCRX_EL2.FnXS == '1' then
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Any, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL2 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                    TLBIlevel_Any, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                    TLBIlevel_Any, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL3 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                    TLBIlevel_Any, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                    TLBIlevel_Any, TLBI_AllAttr, X[t]);

```

TLBI RVAAE1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0110 | 0b011 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIRVAAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBILevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBILevel_Any, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
            TLBILevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBILevel_Any, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                TLBILevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBILevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAAE1IS, TLBI RVAAE1ISNXS, TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable

The TLBI RVAAE1IS, TLBI RVAAE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 0000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:

- If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000.
- If TTL==10 and BaseADDR[28:16] is not equal to 00000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVAAE1IS, TLBI RVAAE1ISNXS are UNDEFINED.

Attributes

TLBI RVAAE1IS, TLBI RVAAE1ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAAE1IS, TLBI RVAAE1ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAAE1IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b000 | 0b1000 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVAAE1IS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        elsif PSTATE.EL == EL2 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
            else
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL3 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI RVAAE1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0010 | 0b011 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIRVAAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAAE1OS, TLBI RVAAE1OSNXS, TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable

The TLBI RVAAE1OS, TLBI RVAAE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 0000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:

TLBI RVAAE1OS, TLBI RVAAE1OSNXS, TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable

- If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000.
- If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI RVAAE1OS, TLBI RVAAE1OSNXS are UNDEFINED.

Attributes

TLBI RVAAE1OS, TLBI RVAAE1OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAAE1OS, TLBI RVAAE1OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAAE1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b000 | 0b1000 | 0b0101 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVAAE1OS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        elsif PSTATE.EL == EL2 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
            else
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL3 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI RVAAE1OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0101 | 0b011 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIRVAAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAALE1, TLBI RVAALE1NXS, TLB Range Invalidate by VA, All ASID, Last level, EL1

The TLBI RVAALE1, TLBI RVAALE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVAALE1, TLBI RVAALE1NXS are UNDEFINED.

Attributes

TLBI RVAALE1, TLBI RVAALE1NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|----|-----|----|----------|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAALE1, TLBI RVAALE1NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAALE1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0110 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVAALE1 == '1'
    then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
        then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Last, TLBI_AllAttr, X[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
            && HCRX_EL2.FnXS == '1' then
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Last, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL2 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                    TLBIlevel_Last, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                    TLBIlevel_Last, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL3 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                    TLBIlevel_Last, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                    TLBIlevel_Last, TLBI_AllAttr, X[t]);

```

TLBI RVAALE1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0110 | 0b111 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIRVAALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAALE1IS, TLBI RVAALE1ISNXS, TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

The TLBI RVAALE1IS, TLBI RVAALE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.

- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 00000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVAALE1IS, TLBI RVAALE1ISNXS are UNDEFINED.

Attributes

TLBI RVAALE1IS, TLBI RVAALE1ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]
When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAALE1IS, TLBI RVAALE1ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAALE1IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0010 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVAALE1IS ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);

```

TLBI RVAALE1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0010 | 0b111 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIRVAALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAALE1OS, TLBI RVAALE1OSNXS, TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

The TLBI RVAALE1OS, TLBI RVAALE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.

- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 00000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI RVAALE1OS, TLBI RVAALE1OSNXS are UNDEFINED.

Attributes

TLBI RVAALE1OS, TLBI RVAALE1OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]
When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAALE1OS, TLBI RVAALE1OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAALE1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0101 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVAALE10S ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        endif PSTATE.EL == EL2 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
            else
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
            endif PSTATE.EL == EL3 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
                endif
            endif
        endif
    endif
endif

```

TLBI RVAALE10SNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0101 | 0b111 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
&& (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIRVAALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE1, TLBI RVAE1NXS, TLB Range Invalidate by VA, EL1

The TLBI RVAE1, TLBI RVAE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
 - The entry is from a level of lookup above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVAE1, TLBI RVAE1NXS are UNDEFINED.

Attributes

TLBI RVAE1, TLBI RVAE1NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| ASID | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|--|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE1, TLBI RVAE1NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAE1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVAE1 == '1'
    then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
        then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Any, TLBI_AllAttr, X[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
            && HCRX_EL2.FnXS == '1' then
                AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Any, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL2 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                    TLBIlevel_Any, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                    TLBIlevel_Any, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL3 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                    TLBIlevel_Any, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                    TLBIlevel_Any, TLBI_AllAttr, X[t]);

```

TLBI RVAE1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0110 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIRVAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE1IS, TLBI RVAE1ISNXS, TLB Range Invalidate by VA, EL1, Inner Shareable

The TLBI RVAE1IS, TLBI RVAE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
 - The entry is from a level of lookup above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.

- For the 64K translation granule:
 - If $TTL == 01$ and $BaseADDR[41:16]$ is not equal to 000000000000000000000000.
 - If $TTL == 10$ and $BaseADDR[28:16]$ is not equal to 0000000000000000.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVAE1IS, TLBI RVAE1ISNXS are UNDEFINED.

Attributes

TLBI RVAE1IS, TLBI RVAE1ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE1IS, TLBI RVAE1ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAE1IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b000 | 0b1000 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVAE1IS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        elsif PSTATE.EL == EL2 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
            else
                AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL3 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI RVAE1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0010 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIRVAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE1OS, TLBI RVAE1OSNXS, TLB Range Invalidate by VA, EL1, Outer Shareable

The TLBI RVAE1OS, TLBI RVAE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
 - The entry is from a level of lookup above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 0000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.

- For the 64K translation granule:
 - If $TTL == 01$ and $BaseADDR[41:16]$ is not equal to 000000000000000000000000.
 - If $TTL == 10$ and $BaseADDR[28:16]$ is not equal to 00000000000000.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI RVAE1OS, TLBI RVAE1OSNXS are UNDEFINED.

Attributes

TLBI RVAE1OS, TLBI RVAE1OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE1OS, TLBI RVAE1OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAE1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b000 | 0b1000 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVAE10S == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        elsif PSTATE.EL == EL2 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
            else
                AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL3 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI RVAE10SNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0101 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
&& (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIRVAE10S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE2, TLBI RVAE2NXS, TLB Range Invalidate by VA, EL2

The TLBI RVAE2, TLBI RVAE2NXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA in the specified range determined by the formula $[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) * 2^{(5 * \text{SCALE} + 1)} * \text{Translation_Granule_Size})]$ using the EL2 or EL2&0 translation regime for the Security state.
- If [HCR_EL2.E2H](#) == 0, the entry is from any level of the translation table walk.
- If [HCR_EL2.E2H](#) == 1, one of the following applies:
 - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of the translation table walk.
 - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of [SCR_EL3.NS](#) if FEAT_RME is not implemented, or [SCR_EL3.{NSE, NS}](#) if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

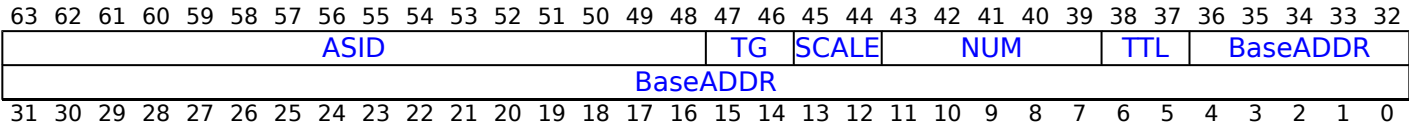
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVAE2, TLBI RVAE2NXS are UNDEFINED.

Attributes

TLBI RVAE2, TLBI RVAE2NXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]
When HCR_EL2.E2H == 1:

- ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.
- Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.
- If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL2.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE2, TLBI RVAE2NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAE2{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
elseif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elseif HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);

```

TLBI RVAE2NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0110 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE2IS, TLBI RVAE2ISNXS, TLB Range Invalidate by VA, EL2, Inner Shareable

The TLBI RVAE2IS, TLBI RVAE2ISNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA in the specified range determined by the formula $[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) * 2^{(5 * \text{SCALE} + 1)} * \text{Translation_Granule_Size})]$ using the EL2 or EL2&0 translation regime for the Security state.
- If [HCR_EL2.E2H](#) == 0, the entry is from any level of the translation table walk.
- If [HCR_EL2.E2H](#) == 1, one of the following applies:
 - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of the translation table walk.
 - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of [SCR_EL3.NS](#) if FEAT_RME is not implemented, or [SCR_EL3.{NSE, NS}](#) if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If $\text{TTL} == 01$ and $\text{BaseADDR}[29:12]$ is not equal to 00000000000000000000.
 - If $\text{TTL} == 10$ and $\text{BaseADDR}[20:12]$ is not equal to 000000000.
- For the 16K translation granule:
 - If $\text{TTL} == 10$ and $\text{BaseADDR}[24:14]$ is not equal to 000000000000.
- For the 64K translation granule:
 - If $\text{TTL} == 01$ and $\text{BaseADDR}[41:16]$ is not equal to 00000000000000000000000000000000.
 - If $\text{TTL} == 10$ and $\text{BaseADDR}[28:16]$ is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

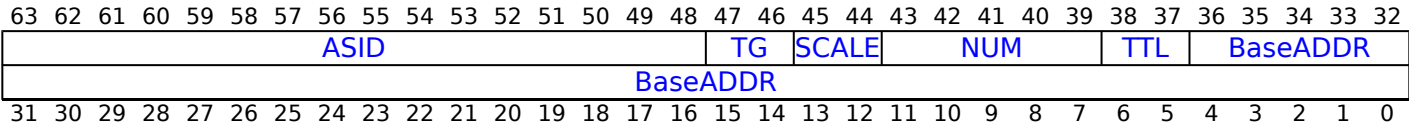
Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVAE2IS, TLBI RVAE2ISNXS are UNDEFINED.

Attributes

TLBI RVAE2IS, TLBI RVAE2ISNXS is a 64-bit System instruction.

Field descriptions



ASID, bits [63:48]
When HCR_EL2.E2H == 1:

- ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.
- Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.
- If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

- Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|--|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL2.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE2IS, TLBI RVAE2ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAE2IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);

```

TLBI RVAE2ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0010 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE2OS, TLBI RVAE2OSNXS, TLB Range Invalidate by VA, EL2, Outer Shareable

The TLBI RVAE2OS, TLBI RVAE2OSNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA in the specified range determined by the formula $[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) * 2^{(5 * \text{SCALE} + 1)} * \text{Translation_Granule_Size})]$ using the EL2 or EL2&0 translation regime for the Security state.
- If [HCR_EL2.E2H](#) == 0, the entry is from any level of the translation table walk.
- If [HCR_EL2.E2H](#) == 1, one of the following applies:
 - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of the translation table walk.
 - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of [SCR_EL3.NS](#) if FEAT_RME is not implemented, or [SCR_EL3.{NSE, NS}](#) if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI RVAE2OS, TLBI RVAE2OSNXS are UNDEFINED.

Attributes

TLBI RVAE2OS, TLBI RVAE2OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|--|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL2.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE2OS, TLBI RVAE2OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAE2OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);

```

TLBI RVAE2OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0101 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE3, TLBI RVAE3NXS, TLB Range Invalidate by VA, EL3

The TLBI RVAE3, TLBI RVAE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula $[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) * 2^{(5 * \text{SCALE} + 1)} * \text{Translation_Granule_Size})]$.

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If $\text{TTL} == 01$ and $\text{BaseADDR}[29:12]$ is not equal to 00000000000000000000.
 - If $\text{TTL} == 10$ and $\text{BaseADDR}[20:12]$ is not equal to 000000000.
- For the 16K translation granule:
 - If $\text{TTL} == 10$ and $\text{BaseADDR}[24:14]$ is not equal to 000000000000.
- For the 64K translation granule:
 - If $\text{TTL} == 01$ and $\text{BaseADDR}[41:16]$ is not equal to 00000000000000000000000000000000.
 - If $\text{TTL} == 10$ and $\text{BaseADDR}[28:16]$ is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVAE3, TLBI RVAE3NXS are UNDEFINED.

Attributes

TLBI RVAE3, TLBI RVAE3NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|--|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL3.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE3, TLBI RVAE3NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAE3{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1000 | 0b0110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_NSH, TLBIlevel_Any,
    TLBI_AllAttr, X[t]);

```

TLBI RVAE3NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0110 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_NSH, TLBIlevel_Any,
    TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE3IS, TLBI RVAE3ISNXS, TLB Range Invalidate by VA, EL3, Inner Shareable

The TLBI RVAE3IS, TLBI RVAE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If $TTL == 01$ and $BaseADDR[29:12]$ is not equal to 000000000000000000.
 - If $TTL == 10$ and $BaseADDR[20:12]$ is not equal to 000000000.
- For the 16K translation granule:
 - If $TTL == 10$ and $BaseADDR[24:14]$ is not equal to 00000000000.
- For the 64K translation granule:
 - If $TTL == 01$ and $BaseADDR[41:16]$ is not equal to 00000000000000000000000000000000.
 - If $TTL == 10$ and $BaseADDR[28:16]$ is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVAE3IS, TLBI RVAE3ISNXS are UNDEFINED.

Attributes

TLBI RVAE3IS, TLBI RVAE3ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | | | |
| | | | | | | | | | | | | | | | | BaseADDR | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|--|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL3.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE3IS, TLBI RVAE3ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAE3IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1000 | 0b0010 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_ISH, TLBIlevel_Any,
    TLBI_AllAttr, X[t]);
```

TLBI RVAE3ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0010 | 0b001 |

```
if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_ISH, TLBIlevel_Any,
    TLBI_ExcludeXS, X[t]);
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVAE3OS, TLBI RVAE3OSNXS, TLB Range Invalidate by VA, EL3, Outer Shareable

The TLBI RVAE3OS, TLBI RVAE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If $TTL == 01$ and $BaseADDR[29:12]$ is not equal to 000000000000000000.
 - If $TTL == 10$ and $BaseADDR[20:12]$ is not equal to 000000000.
- For the 16K translation granule:
 - If $TTL == 10$ and $BaseADDR[24:14]$ is not equal to 00000000000.
- For the 64K translation granule:
 - If $TTL == 01$ and $BaseADDR[41:16]$ is not equal to 00000000000000000000000000000000.
 - If $TTL == 10$ and $BaseADDR[28:16]$ is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI RVAE3OS, TLBI RVAE3OSNXS are UNDEFINED.

Attributes

TLBI RVAE3OS, TLBI RVAE3OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | | | |
| | | | | | | | | | | | | | | | | BaseADDR | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|--|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL3.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVAE3OS, TLBI RVAE3OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVAE3OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1000 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_0SH, TLBIlevel_Any,
    TLBI_AllAttr, X[t]);

```

TLBI RVAE3OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0101 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_0SH, TLBIlevel_Any,
    TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE1, TLBI RVALE1NXS, TLB Range Invalidate by VA, Last level, EL1

The TLBI RVALE1, TLBI RVALE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

For more information about the architectural requirements for this System instruction, see 'Invalidation of TLB entries from stage 2 translations'.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVALE1, TLBI RVALE1NXS are UNDEFINED.

Attributes

TLBI RVALE1, TLBI RVALE1NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|----|-----|----|----------|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| ASID | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE1, TLBI RVALE1NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVALE1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b000 | 0b1000 | 0b0110 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVALE1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
                AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL2 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
                elsif PSTATE.EL == EL3 then
                    if HCR_EL2.<E2H,TGE> == '11' then
                        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
                    else
                        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);

```

TLBI RVALE1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0110 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIRVALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            end
        end
    end

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE1IS, TLBI RVALE1ISNXS, TLB Range Invalidate by VA, Last level, EL1, Inner Shareable

The TLBI RVALE1IS, TLBI RVALE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 0000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:

- If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000.
- If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVALE1IS, TLBI RVALE1ISNXS are UNDEFINED.

Attributes

TLBI RVALE1IS, TLBI RVALE1ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------------|--|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE1IS, TLBI RVALE1ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVALE1IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b000 | 0b1000 | 0b0010 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVALE1IS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);

```

TLBI RVALE1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0010 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIRVALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE1OS, TLBI RVALE1OSNXS, TLB Range Invalidate by VA, Last level, EL1, Outer Shareable

The TLBI RVALE1OS, TLBI RVALE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- The entry is within the address range determined by the formula $[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) * 2^{(5 * \text{SCALE} + 1)} * \text{Translation_Granule_Size})]$.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 0000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:

- If TTL==01 and BaseADDR[41:16] is not equal to 000000000000000000000000.
- If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI RVALE1OS, TLBI RVALE1OSNXS are UNDEFINED.

Attributes

TLBI RVALE1OS, TLBI RVALE1OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL1.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE1OS, TLBI RVALE1OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVALE1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b000 | 0b1000 | 0b0101 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIRVALE10S == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);

```

TLBI RVALE10SNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0101 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIRVALE10S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_RVA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE2, TLBI RVALE2NXS, TLB Range Invalidate by VA, Last level, EL2

The TLBI RVALE2, TLBI RVALE2NXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA in the specified range determined by the formula $[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) * 2^{(5 * \text{SCALE} + 1)} * \text{Translation_Granule_Size})]$ using the EL2 or EL2&0 translation regime for the Security state.
- If [HCR_EL2.E2H](#) == 0, the entry is from the final level of the translation table walk.
- If [HCR_EL2.E2H](#) == 1, one of the following applies:
 - The entry is a global entry from the final level of the translation table walk.
 - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of [SCR_EL3.NS](#) if FEAT_RME is not implemented, or [SCR_EL3.{NSE, NS}](#) if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVALE2, TLBI RVALE2NXS are UNDEFINED.

Attributes

TLBI RVALE2, TLBI RVALE2NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL2.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE2, TLBI RVALE2NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVALE2{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0110 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    
```

TLBI RVALE2NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0110 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE2IS, TLBI RVALE2ISNXS, TLB Range Invalidate by VA, Last level, EL2, Inner Shareable

The TLBI RVALE2IS, TLBI RVALE2ISNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA in the specified range determined by the formula $[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) * 2^{(5 * \text{SCALE} + 1)} * \text{Translation_Granule_Size})]$ using the EL2 or EL2&0 translation regime for the Security state.
- If [HCR_EL2.E2H](#) == 0, the entry is from the final level of the translation table walk.
- If [HCR_EL2.E2H](#) == 1, one of the following applies:
 - The entry is a global entry from the final level of the translation table walk.
 - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of [SCR_EL3.NS](#) if FEAT_RME is not implemented, or [SCR_EL3.{NSE, NS}](#) if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVALE2IS, TLBI RVALE2ISNXS are UNDEFINED.

Attributes

TLBI RVALE2IS, TLBI RVALE2ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL2.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE2IS, TLBI RVALE2ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVALE2IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0010 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    
```

TLBI RVALE2ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0010 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE2OS, TLBI RVALE2OSNXS, TLB Range Invalidate by VA, Last level, EL2, Outer Shareable

The TLBI RVALE2OS, TLBI RVALE2OSNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA in the specified range determined by the formula $[\text{BaseADDR} \leq \text{VA} < \text{BaseADDR} + ((\text{NUM} + 1) * 2^{(5 * \text{SCALE} + 1)} * \text{Translation_Granule_Size})]$ using the EL2 or EL2&0 translation regime for the Security state.
- If [HCR_EL2.E2H](#) == 0, the entry is from the final level of the translation table walk.
- If [HCR_EL2.E2H](#) == 1, one of the following applies:
 - The entry is a global entry from the final level of the translation table walk.
 - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of [SCR_EL3.NS](#) if FEAT_RME is not implemented, or [SCR_EL3.{NSE, NS}](#) if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If TTL==01 and BaseADDR[29:12] is not equal to 00000000000000000000.
 - If TTL==10 and BaseADDR[20:12] is not equal to 000000000.
- For the 16K translation granule:
 - If TTL==10 and BaseADDR[24:14] is not equal to 000000000000.
- For the 64K translation granule:
 - If TTL==01 and BaseADDR[41:16] is not equal to 00000000000000000000000000000000.
 - If TTL==10 and BaseADDR[28:16] is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI RVALE2OS, TLBI RVALE2OSNXS are UNDEFINED.

Attributes

TLBI RVALE2OS, TLBI RVALE2OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----|----------|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | | BaseADDR | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

When HCR_EL2.E2H == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|---|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]**When FEAT_LPA2 is implemented and TCR_EL2.DS == 1:**

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE2OS, TLBI RVALE2OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVALE2OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0101 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);

```

TLBI RVALE2OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0101 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_RVA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE3, TLBI RVALE3NXS, TLB Range Invalidate by VA, Last level, EL3

The TLBI RVALE3, TLBI RVALE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.

The invalidation applies to the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If $TTL == 01$ and $BaseADDR[29:12]$ is not equal to 00000000000000000000.
 - If $TTL == 10$ and $BaseADDR[20:12]$ is not equal to 000000000.
- For the 16K translation granule:
 - If $TTL == 10$ and $BaseADDR[24:14]$ is not equal to 000000000000.
- For the 64K translation granule:
 - If $TTL == 01$ and $BaseADDR[41:16]$ is not equal to 00000000000000000000000000000000.
 - If $TTL == 10$ and $BaseADDR[28:16]$ is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVALE3, TLBI RVALE3NXS are UNDEFINED.

Attributes

TLBI RVALE3, TLBI RVALE3NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|--|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL3.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE3, TLBI RVALE3NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVALE3{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1000 | 0b0110 | 0b101 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_NSH, TLBIlevel_Last,
    TLBI_AllAttr, X[t]);
```

TLBI RVALE3NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0110 | 0b101 |

```
if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_NSH, TLBIlevel_Last,
    TLBI_ExcludeXS, X[t]);
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE3IS, TLBI RVALE3ISNXS, TLB Range Invalidate by VA, Last level, EL3, Inner Shareable

The TLBI RVALE3IS, TLBI RVALE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If $TTL == 01$ and $BaseADDR[29:12]$ is not equal to 000000000000000000.
 - If $TTL == 10$ and $BaseADDR[20:12]$ is not equal to 000000000.
- For the 16K translation granule:
 - If $TTL == 10$ and $BaseADDR[24:14]$ is not equal to 00000000000.
- For the 64K translation granule:
 - If $TTL == 01$ and $BaseADDR[41:16]$ is not equal to 00000000000000000000000000000000.
 - If $TTL == 10$ and $BaseADDR[28:16]$ is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented. Otherwise, direct accesses to TLBI RVALE3IS, TLBI RVALE3ISNXS are UNDEFINED.

Attributes

TLBI RVALE3IS, TLBI RVALE3ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|-------|-----|----|----|----|-----|----|----------|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | TTL | | BaseADDR | | | | | | | | | |
| | | | | | | | | | | | | | | | | BaseADDR | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|--|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL3.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE3IS, TLBI RVALE3ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVALE3IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1000 | 0b0010 | 0b101 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_ISH, TLBIlevel_Last,
    TLBI_AllAttr, X[t]);
```

TLBI RVALE3ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0010 | 0b101 |

```
if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_ISH, TLBIlevel_Last,
    TLBI_ExcludeXS, X[t]);
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI RVALE3OS, TLBI RVALE3OSNXS, TLB Range Invalidate by VA, Last level, EL3, Outer Shareable

The TLBI RVALE3OS, TLBI RVALE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.
- The entry is within the address range determined by the formula $[BaseADDR \leq VA < BaseADDR + ((NUM + 1) * 2^{(5 * SCALE + 1)} * Translation_Granule_Size)]$.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

The range of addresses invalidated is UNPREDICTABLE when:

- For the 4K translation granule:
 - If $TTL == 01$ and $BaseADDR[29:12]$ is not equal to 000000000000000000.
 - If $TTL == 10$ and $BaseADDR[20:12]$ is not equal to 000000000.
- For the 16K translation granule:
 - If $TTL == 10$ and $BaseADDR[24:14]$ is not equal to 00000000000.
- For the 64K translation granule:
 - If $TTL == 01$ and $BaseADDR[41:16]$ is not equal to 00000000000000000000000000000000.
 - If $TTL == 10$ and $BaseADDR[28:16]$ is not equal to 0000000000000000.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIRANGE is implemented and FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI RVALE3OS, TLBI RVALE3OSNXS are UNDEFINED.

Attributes

TLBI RVALE3OS, TLBI RVALE3OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|-----|----|----|----|----|----|-----|----------|----|----|----|----|----|----|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | |
| RES0 | | | | | | | | | | | | | | | | TG | SCALE | NUM | | | | | | TTL | BaseADDR | | | | | | | | | |
| BaseADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | |

Bits [63:48]

Reserved, RES0.

TG, bits [47:46]

Translation granule size.

| TG | Meaning |
|------|--------------------------|
| 0b00 | Reserved. |
| 0b01 | 4K translation granule. |
| 0b10 | 16K translation granule. |
| 0b11 | 64K translation granule. |

The instruction takes a translation granule size for the translations that are being invalidated. If the translations used a different translation granule size than the one being specified, then the architecture does not require that the instruction invalidates any entries.

SCALE, bits [45:44]

The exponent element of the calculation that is used to produce the upper range.

NUM, bits [43:39]

The base element of the calculation that is used to produce the upper range.

TTL, bits [38:37]

TTL Level hint. The TTL hint is only guaranteed to invalidate entries in the range that match the level described by the TTL hint.

| TTL | Meaning |
|------|--|
| 0b00 | The entries in the range can be using any level for the translation table entries. |
| 0b01 | All entries to invalidate are Level 1 translation table entries. If FEAT_LPA2 is not implemented, when using a 16KB translation granule, this value is reserved and hardware should treat this field as 0b00. |
| 0b10 | All entries to invalidate are Level 2 translation table entries. |
| 0b11 | All entries to invalidate are Level 3 translation table entries. |

BaseADDR, bits [36:0]

When FEAT_LPA2 is implemented and TCR_EL3.DS == 1:

The starting address for the range of the maintenance instructions. This field is BaseADDR[52:16] for all translation granules.

When using a 4KB translation granule, BaseADDR[15:12] is treated as 0b0000.

When using a 16KB translation granule, BaseADDR[15:14] is treated as 0b00.

Otherwise:

The starting address for the range of the maintenance instruction.

When using a 4KB translation granule, this field is BaseADDR[48:12].

When using a 16KB translation granule, this field is BaseADDR[50:14].

When using a 64KB translation granule, this field is BaseADDR[52:16].

Executing the TLBI RVALE3OS, TLBI RVALE3OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI RVALE3OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1000 | 0b0101 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_0SH, TLBIlevel_Last,
    TLBI_AllAttr, X[t]);

```

TLBI RVALE3OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0101 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_RVA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_0SH, TLBIlevel_Last,
    TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAAE1, TLBI VAAE1NXS, TLB Invalidate by VA, All ASID, EL1

The TLBI VAAE1, TLBI VAAE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VAAE1, TLBI VAAE1NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]**When FEAT_TTL is implemented:**

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|------------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAAE1, TLBI VAAE1NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAAE1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b000 | 0b1000 | 0b0111 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVAAE1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
                AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL2 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
                elsif PSTATE.EL == EL3 then
                    if HCR_EL2.<E2H,TGE> == '11' then
                        AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
                    else
                        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI VAAE1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0111 | 0b011 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVAAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAAE1IS, TLBI VAAE1ISNXS, TLB Invalidate by VA, All ASID, EL1, Inner Shareable

The TLBI VAAE1IS, TLBI VAAE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VAAE1IS, TLBI VAAE1ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAAE1IS, TLBI VAAE1ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAAE1IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0011 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVAAE1IS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI VAAE1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0011 | 0b011 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVAAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAAE1OS, TLBI VAAE1OSNXS, TLB Invalidate by VA, All ASID, EL1, Outer Shareable

The TLBI VAAE1OS, TLBI VAAE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI VAAE1OS, TLBI VAAE1OSNXS are UNDEFINED.

Attributes

TLBI VAAE1OS, TLBI VAAE1OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAAE1OS, TLBI VAAE1OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAAE1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0001 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVAAE1OS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI VAAE1OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0001 | 0b011 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVAAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAALE1, TLBI VAALE1NXS, TLB Invalidate by VA, All ASID, Last level, EL1

The TLBI VAALE1, TLBI VAALE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VAALE1, TLBI VAALE1NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]**When FEAT_TTL is implemented:**

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|------------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAALE1, TLBI VAALE1NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAALE1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b000 | 0b1000 | 0b0111 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVAALe1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
                AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL2 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL3 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);

```

TLBI VAALE1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0111 | 0b111 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVAALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAALE1IS, TLBI VAALE1ISNXS, TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable

The TLBI VAALE1IS, TLBI VAALE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VAALE1IS, TLBI VAALE1ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAALE1IS, TLBI VAALE1ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAALE1IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0011 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVAALE1IS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);

```

TLBI VAALE1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0011 | 0b111 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
&& (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVAALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAALE1OS, TLBI VAALE1OSNXS, TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable

The TLBI VAALE1OS, TLBI VAALE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

Note

For the EL1&0 and EL2&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI VAALE1OS, TLBI VAALE1OSNXS are UNDEFINED.

Attributes

TLBI VAALE1OS, TLBI VAALE1OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAALE1OS, TLBI VAALE1OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAALE1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0001 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVAALE1OS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);

```

TLBI VAALE1OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0001 | 0b111 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
&& (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVAALE10S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VAA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE1, TLBI VAE1NXS, TLB Invalidate by VA, EL1

The TLBI VAE1, TLBI VAE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
 - The entry is from a level of lookup above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VAE1, TLBI VAE1NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|------------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE1, TLBI VAE1NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAE1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0111 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
        then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
                AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL2 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL3 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI VAE1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0111 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVAE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE1IS, TLBI VAE1ISNXS, TLB Invalidate by VA, EL1, Inner Shareable

The TLBI VAE1IS, TLBI VAE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
 - The entry is from a level of lookup above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VAE1IS, TLBI VAE1ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| ASID | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE1IS, TLBI VAE1ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAE1IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVAE1IS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI VAE1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0011 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVAE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE1OS, TLBI VAE1OSNXS, TLB Invalidate by VA, EL1, Outer Shareable

The TLBI VAE1OS, TLBI VAE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
 - The entry is from a level of lookup above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI VAE1OS, TLBI VAE1OSNXS are UNDEFINED.

Attributes

TLBI VAE1OS, TLBI VAE1OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| ASID | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE1OS, TLBI VAE1OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAE1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVAE1OS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI VAE1OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0001 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVAE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
            TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
                TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE2, TLBI VAE2NXS, TLB Invalidate by VA, EL2

The TLBI VAE2, TLBI VAE2NXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be required to translate the specified VA using the EL2 or EL2&0 translation regime for the Security state.
- If [HCR_EL2.E2H](#) == 0, the entry is from any level of the translation table walk.
- If [HCR_EL2.E2H](#) == 1, one of the following applies:
 - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of the translation table walk.
 - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of [SCR_EL3.NS](#) if FEAT_RME is not implemented, or [SCR_EL3.{NSE, NS}](#) if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VAE2, TLBI VAE2NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE2, TLBI VAE2NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAE2{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0111 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBILevel_Any, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBILevel_Any, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBILevel_Any, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBILevel_Any, TLBI_AllAttr, X[t]);

```

TLBI VAE2NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0111 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBILevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBILevel_Any, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBILevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBILevel_Any, TLBI_ExcludeXS, X[t]);

```


TLBI VAE2IS, TLBI VAE2ISNXS, TLB Invalidate by VA, EL2, Inner Shareable

The TLBI VAE2IS, TLBI VAE2ISNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be required to translate the specified VA using the EL2 or EL2&0 translation regime for the Security state.
- If [HCR_EL2.E2H](#) == 0, the entry is from any level of the translation table walk.
- If [HCR_EL2.E2H](#) == 1, one of the following applies:
 - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of the translation table walk.
 - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of [SCR_EL3.NS](#) if FEAT_RME is not implemented, or [SCR_EL3.{NSE, NS}](#) if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VAE2IS, TLBI VAE2ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|------------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE2IS, TLBI VAE2ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAE2IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
    
```

TLBI VAE2ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0011 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    
```


TLBI VAE2OS, TLBI VAE2OSNXS, TLB Invalidate by VA, EL2, Outer Shareable

The TLBI VAE2OS, TLBI VAE2OSNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be required to translate the specified VA using the EL2 or EL2&0 translation regime for the Security state.
- If [HCR_EL2.E2H](#) == 0, the entry is from any level of the translation table walk.
- If [HCR_EL2.E2H](#) == 1, one of the following applies:
 - The entry is from a level of the translation table walk above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of the translation table walk.
 - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of [SCR_EL3.NS](#) if FEAT_RME is not implemented, or [SCR_EL3.{NSE, NS}](#) if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI VAE2OS, TLBI VAE2OSNXS are UNDEFINED.

Attributes

TLBI VAE2OS, TLBI VAE2OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| ASID | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

ASID, bits [63:48]**When HCR_EL2.E2H == 1:**

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TTL, bits [47:44]**When FEAT_TTL is implemented:**

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|------------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE2OS, TLBI VAE2OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAE2OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Any, TLBI_AllAttr, X[t]);

```

TLBI VAE2OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0001 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Any, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE3, TLBI VAE3NXS, TLB Invalidate by VA, EL3

The TLBI VAE3, TLBI VAE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VAE3, TLBI VAE3NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | VA[55:12] | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|------------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE3, TLBI VAE3NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAE3{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b110 | 0b1000 | 0b0111 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_NSH, TLBILevel_Any,
    TLBI_AllAttr, X[t]);

```

TLBI VAE3NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0111 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_NSH, TLBILevel_Any,
    TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE3IS, TLBI VAE3ISNXS, TLB Invalidate by VA, EL3, Inner Shareable

The TLBI VAE3IS, TLBI VAE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VAE3IS, TLBI VAE3ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|------------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE3IS, TLBI VAE3ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAE3IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b110 | 0b1000 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_ISH, TLBILevel_Any,
    TLBI_AllAttr, X[t]);

```

TLBI VAE3ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0011 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_ISH, TLBILevel_Any,
    TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VAE3OS, TLBI VAE3OSNXS, TLB Invalidate by VA, EL3, Outer Shareable

The TLBI VAE3OS, TLBI VAE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI VAE3OS, TLBI VAE3OSNXS are UNDEFINED.

Attributes

TLBI VAE3OS, TLBI VAE3OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|------------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VAE3OS, TLBI VAE3OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VAE3OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b110 | 0b1000 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_OSH, TLBILevel_Any,
    TLBI_AllAttr, X[t]);

```

TLBI VAE3OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0001 | 0b001 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_OSH, TLBILevel_Any,
    TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE1, TLBI VALE1NXS, TLB Invalidate by VA, Last level, EL1

The TLBI VALE1, TLBI VALE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VALE1, TLBI VALE1NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| ASID | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|------------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE1, TLBI VALE1NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VALE1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0111 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVALE1 == '1'
    then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
        then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
                AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL2 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
            elsif PSTATE.EL == EL3 then
                if HCR_EL2.<E2H,TGE> == '11' then
                    AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
                else
                    AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);

```

TLBI VALE1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0111 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVALE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE1IS, TLBI VALE1ISNXS, TLB Invalidate by VA, Last level, EL1, Inner Shareable

The TLBI VALE1IS, TLBI VALE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VALE1IS, TLBI VALE1ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| ASID | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE1IS, TLBI VALE1ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VALE1IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0011 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVALE1IS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, X[t]);

```

TLBI VALE1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0011 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVALE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE1OS, TLBI VALE1OSNXS, TLB Invalidate by VA, Last level, EL1, Outer Shareable

The TLBI VALE1OS, TLBI VALE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA, and one of the following applies:
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI VALE1OS, TLBI VALE1OSNXS are UNDEFINED.

Attributes

TLBI VALE1OS, TLBI VALE1OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| ASID | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE1OS, TLBI VALE1OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VALE1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVALE1OS == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBILevel_Last, TLBI_AllAttr, X[t]);

```

TLBI VALE1OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0001 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVALE1OS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        else
            AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
            TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
            else
                AArch64.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
                TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE2, TLBI VALE2NXS, TLB Invalidate by VA, Last level, EL2

The TLBI VALE2, TLBI VALE2NXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA using the EL2 or EL2&0 translation regime for the Security state.
- If [HCR_EL2.E2H](#) == 0, the entry is from the final level of the translation table walk.
- If [HCR_EL2.E2H](#) == 1, one of the following applies:
 - The entry is a global entry from the final level of the translation table walk.
 - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of [SCR_EL3.NS](#) if FEAT_RME is not implemented, or [SCR_EL3.{NSE, NS}](#) if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VALE2, TLBI VALE2NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | VA[55:12] | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

When [HCR_EL2.E2H](#) == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE2, TLBI VALE2NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VALE2{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0111 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);

```

TLBI VALE2NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0111 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_NSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```


TLBI VALE2IS, TLBI VALE2ISNXS, TLB Invalidate by VA, Last level, EL2, Inner Shareable

The TLBI VALE2IS, TLBI VALE2ISNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA using the EL2 or EL2&0 translation regime for the Security state.
- If [HCR_EL2.E2H](#) == 0, the entry is from the final level of the translation table walk.
- If [HCR_EL2.E2H](#) == 1, one of the following applies:
 - The entry is a global entry from the final level of the translation table walk.
 - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of [SCR_EL3.NS](#) if FEAT_RME is not implemented, or [SCR_EL3.{NSE, NS}](#) if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VALE2IS, TLBI VALE2ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | VA[55:12] | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE2IS, TLBI VALE2ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VALE2IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0011 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);

```

TLBI VALE2ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0011 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```


TLBI VALE2OS, TLBI VALE2OSNXS, TLB Invalidate by VA, Last level, EL2, Outer Shareable

The TLBI VALE2OS, TLBI VALE2OSNXS characteristics are:

Purpose

When EL2 is implemented and enabled in the current Security state, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified VA using the EL2 or EL2&0 translation regime for the Security state.
- If [HCR_EL2.E2H](#) == 0, the entry is from the final level of the translation table walk.
- If [HCR_EL2.E2H](#) == 1, one of the following applies:
 - The entry is a global entry from the final level of the translation table walk.
 - The entry is a non-global entry from the final level of the translation table walk that matches the specified ASID.

The Security state is indicated by the value of [SCR_EL3.NS](#) if FEAT_RME is not implemented, or [SCR_EL3.{NSE, NS}](#) if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI VALE2OS, TLBI VALE2OSNXS are UNDEFINED.

Attributes

TLBI VALE2OS, TLBI VALE2OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

When [HCR_EL2.E2H](#) == 1:

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

If the implementation supports 16 bits of ASID, then the upper 8 bits of the ASID must be written to 0 by software when the context being invalidated only uses 8 bits.

Otherwise:

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|--------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE2OS, TLBI VALE2OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VALE2OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Last, TLBI_AllAttr, X[t]);

```

TLBI VALE2OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0001 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    elsif HCR_EL2.E2H == '1' then
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);
    else
        AArch64.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID[], Shareability_OSH,
        TLBIlevel_Last, TLBI_ExcludeXS, X[t]);

```


TLBI VALE3, TLBI VALE3NXS, TLB Invalidate by VA, Last level, EL3

The TLBI VALE3, TLBI VALE3NXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VALE3, TLBI VALE3NXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|------------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE3, TLBI VALE3NXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VALE3{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b110 | 0b1000 | 0b0111 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_NSH, TLBILevel_Last,
    TLBI_AllAttr, X[t]);

```

TLBI VALE3NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0111 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_NSH, TLBILevel_Last,
    TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE3IS, TLBI VALE3ISNXS, TLB Invalidate by VA, Last level, EL3, Inner Shareable

The TLBI VALE3IS, TLBI VALE3ISNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VALE3IS, TLBI VALE3ISNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|------------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE3IS, TLBI VALE3ISNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VALE3IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b110 | 0b1000 | 0b0011 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_ISH, TLBILevel_Last,
    TLBI_AllAttr, X[t]);

```

TLBI VALE3ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0011 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_ISH, TLBILevel_Last,
    TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VALE3OS, TLBI VALE3OSNXS, TLB Invalidate by VA, Last level, EL3, Outer Shareable

The TLBI VALE3OS, TLBI VALE3OSNXS characteristics are:

Purpose

If EL3 is implemented, invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified VA using the EL3 translation regime.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI VALE3OS, TLBI VALE3OSNXS are UNDEFINED.

Attributes

TLBI VALE3OS, TLBI VALE3OSNXS is a 64-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | TTL | | | | VA[55:12] | | | | | | | | | | | | | |
| VA[55:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:48]

Reserved, RES0.

TTL, bits [47:44]

When FEAT_TTL is implemented:

Translation Table Level. Indicates the level of the translation table walk that holds the leaf entry for the address being invalidated.

| TTL | Meaning |
|------------|--|
| 0b00xx | No information supplied as to the translation table level. Hardware must assume that the entry can be from any level. In this case, TTL<1:0> is RES0. |
| 0b01xx | The entry comes from a 4KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : If FEAT_LPA2 is implemented, level 0. Otherwise, treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |
| 0b10xx | The entry comes from a 16KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : If FEAT_LPA2 is implemented, level 1. Otherwise, treat as if TTL<3:2> is 0b00. 0b10 : Level 2. 0b11 : Level 3. |
| 0b11xx | The entry comes from a 64KB translation granule. The level of walk for the leaf level 0bxx is encoded as: 0b00 : Reserved. Treat as if TTL<3:2> is 0b00. 0b01 : Level 1. 0b10 : Level 2. 0b11 : Level 3. |

If an incorrect value of the TTL field is specified for the entry being invalidated by the instruction, then no entries are required by the architecture to be invalidated from the TLB.

Otherwise:

Reserved, RES0.

VA[55:12], bits [43:0]

Bits[55:12] of the virtual address to match. Any appropriate TLB entries that match the ASID value (if appropriate) and VA will be affected by this System instruction.

If the TLB maintenance instructions are targeting a translation regime that is using AArch32, and so has a VA of only 32 bits, then the software must treat bits[55:32] as RES0.

The treatment of the low-order bits of this field depends on the translation granule size, as follows:

- Where a 4KB translation granule is being used, all bits are valid and used for the invalidation.
- Where a 16KB translation granule is being used, bits [1:0] of this field are RES0 and ignored when the instruction is executed, because VA[13:12] have no effect on the operation of the instruction.
- Where a 64KB translation granule is being used, bits [3:0] of this field are RES0 and ignored when the instruction is executed, because VA[15:12] have no effect on the operation of the instruction.

Executing the TLBI VALE3OS, TLBI VALE3OSNXS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VALE3OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b01 | 0b110 | 0b1000 | 0b0001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_0SH, TLBIlevel_Last,
    TLBI_AllAttr, X[t]);

```

TLBI VALE3OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b110 | 0b1001 | 0b0001 | 0b101 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    AArch64.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL3, VMID[], Shareability_0SH, TLBIlevel_Last,
    TLBI_ExcludeXS, X[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLE1, TLBI VMALLE1NXS, TLB Invalidate by VMID, All at stage 1, EL1

The TLBI VMALLE1, TLBI VMALLE1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VMALLE1, TLBI VMALLE1NXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI VMALLE1, TLBI VMALLE1NXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.

- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VMALLE1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0111 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVMALLE1 == '1'
    then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
        then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBI_ExcludeXS);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBI_AllAttr);
        else
            if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
            && HCRX_EL2.FnXS == '1' then
                AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBI_ExcludeXS);
            else
                AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBI_AllAttr);
            endif
        endif
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
            TLBI_AllAttr);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBI_AllAttr);
        endif
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
            TLBI_AllAttr);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBI_AllAttr);
        endif
    endif
endif
    
```

TLBI VMALLE1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0111 | 0b000 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVMALLE1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.FB == '1' then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBI_ExcludeXS);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBI_ExcludeXS);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
            TLBI_ExcludeXS);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBI_ExcludeXS);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_NSH,
                TLBI_ExcludeXS);
            else
                AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBI_ExcludeXS);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLE1IS, TLBI VMALLE1ISNXS, TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable

The TLBI VMALLE1IS, TLBI VMALLE1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VMALLE1IS, TLBI VMALLE1ISNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI VMALLE1IS, TLBI VMALLE1ISNXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VMALLE1IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVMALLE1IS ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_ExcludeXS);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_AllAttr);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBI_AllAttr);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_AllAttr);
    elsif PSTATE.EL == EL3 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
TLBI_AllAttr);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_AllAttr);

```

TLBI VMALLE1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0011 | 0b000 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLBIS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVMALLE1IS == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBI_ExcludeXS);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
            TLBI_ExcludeXS);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBI_ExcludeXS);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_ISH,
                TLBI_ExcludeXS);
            else
                AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
                TLBI_ExcludeXS);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLE1OS, TLBI VMALLE1OSNXS, TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable

The TLBI VMALLE1OS, TLBI VMALLE1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- When EL2 is implemented and enabled in the current Security state:
 - If [HCR_EL2](#).{E2H, TGE} is not {1, 1}, the entry would be used with the current VMID and would be required to translate the specified VA using the EL1&0 translation regime for the Security state.
 - If [HCR_EL2](#).{E2H, TGE} is {1, 1}, the entry would be required to translate the specified VA using the EL2&0 translation regime for the Security state.
- When EL2 is not implemented or is disabled in the current Security state, the entry would be required to translate the specified VA using the EL1&0 translation regime for the Security state.

The Security state is indicated by the value of [SCR_EL3](#).NS if FEAT_RME is not implemented, or [SCR_EL3](#).{NSE, NS} if FEAT_RME is implemented.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI VMALLE1OS, TLBI VMALLE1OSNXS are UNDEFINED.

Attributes

TLBI VMALLE1OS, TLBI VMALLE1OSNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI VMALLE1OS, TLBI VMALLE1OSNXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VMALLE1OS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1000 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGITR_EL2.TLBIVMALLE1OS ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && IsHCRXEL2Enabled()
&& HCRX_EL2.FnXS == '1' then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBI_ExcludeXS);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBI_AllAttr);
    endif
elsif PSTATE.EL == EL2 then
    if HCR_EL2.<E2H,TGE> == '11' then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBI_AllAttr);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBI_AllAttr);
    endif
elsif PSTATE.EL == EL3 then
    if HCR_EL2.<E2H,TGE> == '11' then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
TLBI_AllAttr);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
TLBI_AllAttr);
    endif

```

TLBI VMALLE1OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b000 | 0b1001 | 0b0001 | 0b000 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TTLB == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.TTLB0S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && IsFeatureImplemented(FEAT_HCX)
    && (!IsHCRXEL2Enabled() || HCRX_EL2.FGTnXS == '0') && HFGITR_EL2.TLBIVMALLE10S == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
        TLBI_ExcludeXS);
    elsif PSTATE.EL == EL2 then
        if HCR_EL2.<E2H,TGE> == '11' then
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
            TLBI_ExcludeXS);
        else
            AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
            TLBI_ExcludeXS);
        elsif PSTATE.EL == EL3 then
            if HCR_EL2.<E2H,TGE> == '11' then
                AArch64.TLBI_VMALL(SecurityStateAtEL(EL2), Regime_EL20, VMID_NONE, Shareability_OSH,
                TLBI_ExcludeXS);
            else
                AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
                TLBI_ExcludeXS);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLS12E1, TLBI VMALLS12E1NXS, TLB Invalidate by VMID, All at Stage 1 and 2, EL1

The TLBI VMALLS12E1, TLBI VMALLS12E1NXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - If [SCR_EL3](#).{NSE, NS} is {0, 0}, then:
 - The entry would be required to translate an address using the Secure EL1&0 translation regime.
 - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
 - If [SCR_EL3](#).{NSE, NS} is {0, 1}, then:
 - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
 - If Non-secure EL2 is implemented, the entry would be used with the current VMID.
 - If [SCR_EL3](#).{NSE, NS} is {1, 1}, then:
 - The entry would be required to translate an address using the Realm EL1&0 translation regime.
 - The entry would be used with the current VMID.
- If FEAT_RME is not implemented, one of the following applies:
 - If [SCR_EL3](#).NS is 0, then:
 - The entry would be required to translate an address using the Secure EL1&0 translation regime.
 - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
 - If [SCR_EL3](#).NS is 1, then:
 - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
 - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

The invalidation applies to the PE that executes this System instruction.

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VMALLS12E1, TLBI VMALLS12E1NXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI VMALLS12E1, TLBI VMALLS12E1NXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VMALLS12E1{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0111 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBI_AllAttr);
    else
        AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBI_AllAttr);

```

TLBI VMALLS12E1NXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0111 | 0b110 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBI_ExcludeXS);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBI_ExcludeXS);
    else
        AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBI_ExcludeXS);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS, TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable

The TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - If [SCR_EL3](#).{NSE, NS} is {0, 0}, then:
 - The entry would be required to translate an address using the Secure EL1&0 translation regime.
 - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
 - If [SCR_EL3](#).{NSE, NS} is {0, 1}, then:
 - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
 - If Non-secure EL2 is implemented, the entry would be used with the current VMID.
 - If [SCR_EL3](#).{NSE, NS} is {1, 1}, then:
 - The entry would be required to translate an address using the Realm EL1&0 translation regime.
 - The entry would be used with the current VMID.
- If FEAT_RME is not implemented, one of the following applies:
 - If [SCR_EL3](#).NS is 0, then:
 - The entry would be required to translate an address using the Secure EL1&0 translation regime.
 - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
 - If [SCR_EL3](#).NS is 1, then:
 - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
 - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Note

From Armv8.4, when a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS, TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

There are no configuration notes.

Attributes

TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VMALLS12E1IS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0011 | 0b110 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBI_AllAttr);
    else
        AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBI_AllAttr);
```

TLBI VMALLS12E1ISNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0011 | 0b110 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBI_ExcludeXS);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBI_ExcludeXS);
    else
        AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBI_ExcludeXS);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBI VMALLS12E1OS, TLBI VMALLS12E1OSNXS, TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable

The TLBI VMALLS12E1OS, TLBI VMALLS12E1OSNXS characteristics are:

Purpose

Invalidates cached copies of translation table entries from TLBs that meet all the following requirements:

- The entry is a stage 1 or stage 2 translation table entry, from any level of the translation table walk.
- If FEAT_RME is implemented, one of the following applies:
 - If [SCR_EL3](#).{NSE, NS} is {0, 0}, then:
 - The entry would be required to translate an address using the Secure EL1&0 translation regime.
 - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
 - If [SCR_EL3](#).{NSE, NS} is {0, 1}, then:
 - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
 - If Non-secure EL2 is implemented, the entry would be used with the current VMID.
 - If [SCR_EL3](#).{NSE, NS} is {1, 1}, then:
 - The entry would be required to translate an address using the Realm EL1&0 translation regime.
 - The entry would be used with the current VMID.
- If FEAT_RME is not implemented, one of the following applies:
 - If [SCR_EL3](#).NS is 0, then:
 - The entry would be required to translate an address using the Secure EL1&0 translation regime.
 - If FEAT_SEL2 is implemented and enabled, the entry would be used with the current VMID.
 - If [SCR_EL3](#).NS is 1, then:
 - The entry would be required to translate an address using the Non-secure EL1&0 translation regime.
 - If Non-secure EL2 is implemented, the entry would be used with the current VMID.

The invalidation applies to all PEs in the same Outer Shareable shareability domain as the PE that executes this System instruction.

Note

When a TLB maintenance instruction is generated to the Secure EL1&0 translation regime and is defined to pass a VMID argument, or would be defined to pass a VMID argument if [SCR_EL3](#).EEL2==1, then:

- A PE with [SCR_EL3](#).EEL2==1 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==0.
 - A PE with [SCR_EL3](#).EEL2==0 is not architecturally required to invalidate any entries in the Secure EL1&0 translation of a PE in the same required shareability domain with [SCR_EL3](#).EEL2==1.
 - A PE is architecturally required to invalidate all relevant entries in the Secure EL1&0 translation of a System MMU in the same required shareability domain with a VMID of 0.
-

Note

For the EL1&0 translation regimes, the invalidation applies to both global entries and non-global entries with any ASID.

If FEAT_XS is implemented, the nXS variant of this System instruction is defined.

TLBI VMALLS12E1IOS, TLBI VMALLS12E1OSNXS, TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable

Both variants perform the same invalidation, but the TLBI System instruction without the nXS qualifier waits for all memory accesses using in-scope old translation information to complete before it is considered complete.

The TLBI System instruction with the nXS qualifier is considered complete when the subset of these memory accesses with XS attribute set to 0 are complete.

Configuration

This instruction is present only when FEAT_TLBIOS is implemented. Otherwise, direct accesses to TLBI VMALLS12E1IOS, TLBI VMALLS12E1OSNXS are UNDEFINED.

Attributes

TLBI VMALLS12E1IOS, TLBI VMALLS12E1OSNXS is a 64-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Xt> is ignored.

Executing the TLBI VMALLS12E1IOS, TLBI VMALLS12E1OSNXS instruction

The Rt field should be set to 0b11111. If the Rt field is not set to 0b11111, it is CONSTRAINED UNPREDICTABLE whether:

- The instruction is UNDEFINED.
- The instruction behaves as if the Rt field is set to 0b11111.

Accesses to this instruction use the following encodings in the System instruction encoding space:

TLBI VMALLS12E1IOS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1000 | 0b0001 | 0b110 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBI_AllAttr);
    else
        AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBI_AllAttr);
```

TLBI VMALLS12E1OSNXS{, <Xt>}

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b01 | 0b100 | 0b1001 | 0b0001 | 0b110 |

```

if !IsFeatureImplemented(FEAT_XS) then
    UNDEFINED;
elsif PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBI_ExcludeXS);
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        AArch64.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBI_ExcludeXS);
    else
        AArch64.TLBI_VMALLS12(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_OSH,
    TLBI_ExcludeXS);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDR2_EL0, EL0 Read/Write Software Thread ID Register 2

The TPIDR2_EL0 characteristics are:

Purpose

Provides a location where SME-aware software executing at EL0 can store thread identifying information, for context management purposes.

The PE makes no use of this register.

Configuration

This register is present only when FEAT_SME is implemented. Otherwise, direct accesses to TPIDR2_EL0 are UNDEFINED.

Attributes

TPIDR2_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | Thread ID | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | Thread ID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR2_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TPIDR2_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elseif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnTP2 == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.EnTP2 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGTR_EL2.nTPIDR2_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TPIDR2_EL0;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.nTPIDR2_EL0 == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TPIDR2_EL0;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elseif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TPIDR2_EL0;
elseif PSTATE.EL == EL3 then
    return TPIDR2_EL0;
    
```

MSR TPIDR2_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnTP2 == '0' then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.EnTP2 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGWTR_EL2.nTPIDR2_EL0 == '0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TPIDR2_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.nTPIDR2_EL0 == '0'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TPIDR2_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && SCR_EL3.EnTP2 == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && SCR_EL3.EnTP2 == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TPIDR2_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    TPIDR2_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDR_EL0, EL0 Read/Write Software Thread ID Register

The TPIDR_EL0 characteristics are:

Purpose

Provides a location where software executing at EL0 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

Configuration

AArch64 System register TPIDR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [TPIDRURW\[31:0\]](#).

Attributes

TPIDR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | Thread ID | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | Thread ID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TPIDR_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGTR_EL2.TPIDR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TPIDR_EL0;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.TPIDR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TPIDR_EL0;
elsif PSTATE.EL == EL2 then
    return TPIDR_EL0;
elsif PSTATE.EL == EL3 then
    return TPIDR_EL0;

```

MSR TPIDR_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGWTR_EL2.TPIDR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDR_EL0 = X[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.TPIDR_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDR_EL0 = X[t];
elsif PSTATE.EL == EL2 then
    TPIDR_EL0 = X[t];
elsif PSTATE.EL == EL3 then
    TPIDR_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDR_EL1, EL1 Software Thread ID Register

The TPIDR_EL1 characteristics are:

Purpose

Provides a location where software executing at EL1 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

Configuration

AArch64 System register TPIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [TPIDRPRW\[31:0\]](#).

Attributes

TPIDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Thread ID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Thread ID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TPIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b100 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.TPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TPIDR_EL1;
elsif PSTATE.EL == EL2 then
    return TPIDR_EL1;
elsif PSTATE.EL == EL3 then
    return TPIDR_EL1;
```

MSR TPIDR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.TPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    TPIDR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TPIDR_EL1 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDR_EL2, EL2 Software Thread ID Register

The TPIDR_EL2 characteristics are:

Purpose

Provides a location where software executing at EL2 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

Configuration

AArch64 System register TPIDR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HTPIDR\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

TPIDR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Thread ID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Thread ID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TPIDR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1101 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x090];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return TPIDR_EL2;
elsif PSTATE.EL == EL3 then
    return TPIDR_EL2;

```

MSR TPIDR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1101 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x090] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TPIDR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    TPIDR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDR_EL3, EL3 Software Thread ID Register

The TPIDR_EL3 characteristics are:

Purpose

Provides a location where software executing at EL3 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

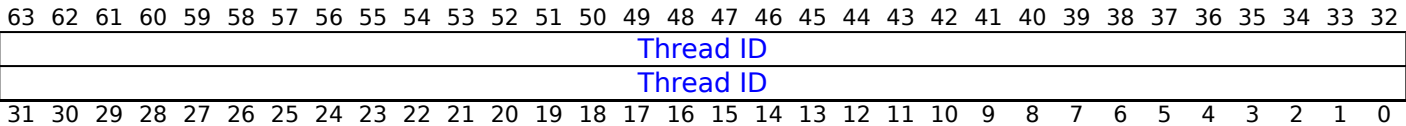
Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to TPIDR_EL3 are UNDEFINED.

Attributes

TPIDR_EL3 is a 64-bit register.

Field descriptions



Bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TPIDR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1101 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return TPIDR_EL3;

```

MSR TPIDR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1101 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TPIDR_EL3 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDRRO_EL0, EL0 Read-Only Software Thread ID Register

The TPIDRRO_EL0 characteristics are:

Purpose

Provides a location where software executing at EL1 or higher can store thread identifying information that is visible to software executing at EL0, for OS management purposes.

The PE makes no use of this register.

Configuration

AArch64 System register TPIDRRO_EL0 bits [31:0] are architecturally mapped to AArch32 System register [TPIDRURO\[31:0\]](#).

Attributes

TPIDRRO_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | Thread ID | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | Thread ID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

Accessing TPIDRRO_EL0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TPIDRRO_EL0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0000 | 0b011 |

```

if PSTATE.EL == EL0 then
    if EL2Enabled() && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') &&
HFGTR_EL2.TPIDRR0_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TPIDRR0_EL0;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.TPIDRR0_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TPIDRR0_EL0;
elseif PSTATE.EL == EL2 then
    return TPIDRR0_EL0;
elseif PSTATE.EL == EL3 then
    return TPIDRR0_EL0;

```

MSR TPIDRR0_EL0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b011 | 0b1101 | 0b0000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.TPIDRR0_EL0 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        TPIDRR0_EL0 = X[t];
elseif PSTATE.EL == EL2 then
    TPIDRR0_EL0 = X[t];
elseif PSTATE.EL == EL3 then
    TPIDRR0_EL0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBBASER_EL1, Trace Buffer Base Address Register

The TRBBASER_EL1 characteristics are:

Purpose

Defines the base address for the trace buffer.

Configuration

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBBASER_EL1 are UNDEFINED.

Attributes

TRBBASER_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | BASE | | | | | | | | | | | | | | | | |
| BASE | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

BASE, bits [63:12]

Trace Buffer Base pointer address. (TRBBASER_EL1.BASE << 12) is the address of the first byte in the trace buffer. Bits [11:0] of the Base pointer address are always zero. If the smallest implemented translation granule is not 4KB, then TRBBASER_EL1[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value Log₂(smallest implemented translation granule).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

Accessing TRBBASER_EL1

The PE might ignore a direct write to TRBBASER_EL1 if [TRBLIMITR_EL1](#).E == 1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRBBASER_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRBBASER_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRBBASER_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRBBASER_EL1;
elsif PSTATE.EL == EL3 then
    return TRBBASER_EL1;

```

MSR TRBBASER_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.TRBBASER_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBBASER_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBBASER_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TRBBASER_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBIDR_EL1, Trace Buffer ID Register

The TRBIDR_EL1 characteristics are:

Purpose

Describes constraints on using the Trace Buffer Unit to software, including whether the Trace Buffer Unit can be programmed at the current Exception level.

Configuration

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBIDR_EL1 are UNDEFINED.

Attributes

TRBIDR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|----|----|-------|----|----|----|----|----|----|----|--|--|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | EA | | | | RES0 | | F | P | Align | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | |

Bits [63:12]

Reserved, RES0.

EA, bits [11:8]

From Armv9.3:

External Abort handling. Describes how the PE manages External aborts on writes made by the Trace Buffer Unit to the trace buffer.

| EA | Meaning |
|--------|---|
| 0b0000 | Not described. |
| 0b0001 | The PE ignores External aborts on writes made by the Trace Buffer Unit. |
| 0b0010 | The External abort generates an SError interrupt at the PE. |

All other values are reserved.

From Armv9.3, the value 0b0000 is not permitted.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

Bits [7:6]

Reserved, RES0.

F, bit [5]

Flag updates. Describes how address translations performed by the Trace Buffer Unit manage the Access flag and dirty state.

| F | Meaning |
|-----|--|
| 0b0 | Hardware management of the Access flag and dirty state for accesses made by the Trace Buffer Unit is always disabled for all translation stages. |
| 0b1 | Hardware management of the Access flag and dirty state for accesses made by the Trace Buffer Unit is controlled in the same way as explicit memory accesses in the trace buffer owning translation regime. |

Note

If hardware management of the Access flag is disabled for a stage of translation, an access to a Page or Block with the Access flag bit not set in the descriptor will generate an Access Flag fault.

If hardware management of the dirty state is disabled for a stage of translation, an access to a Page or Block will ignore the Dirty Bit Modifier in the descriptor and might generate a Permission fault, depending on the values of the access permission bits in the descriptor.

From Armv9.3, the value 0 is not permitted.

Access to this field is **RO**.

P, bit [4]

Programming not allowed. When read at EL3, this field reads as zero. Otherwise, indicates that the trace buffer is owned by a higher Exception level or another Security state. Defined values are:

| P | Meaning |
|-----|--------------------------|
| 0b0 | Programming is allowed. |
| 0b1 | Programming not allowed. |

The value read from this field depends on the current Exception level and the Effective values of [MDCR_EL3.NSTB](#), [MDCR_EL3.NSTBE](#), and [MDCR_EL2.E2TB](#):

- If EL3 is implemented, and the owning Security state is Secure state, this field reads as one from:
 - Non-secure EL1 and Non-secure EL2.
 - If FEAT_RME is implemented, Realm EL1 and Realm EL2.
 - If Secure EL2 is implemented and enabled, and [MDCR_EL2.E2TB](#) is 0b00, Secure EL1.
- If EL3 is implemented, and the owning Security state is Non-secure state, this field reads as one from:
 - Secure EL1.
 - If Secure EL2 is implemented, Secure EL2.
 - If EL2 is implemented and [MDCR_EL2.E2TB](#) is 0b00, Non-secure EL1.
 - If FEAT_RME is implemented, Realm EL1 and Realm EL2.
- If FEAT_RME is implemented, and the owning Security state is Realm state, this field reads as one from:
 - Non-secure EL1 and Non-secure EL2.
 - Secure EL1 and Secure EL2.
 - If [MDCR_EL2.E2TB](#) is 0b00, Realm EL1.
- If EL3 is not implemented, EL2 is implemented, and [MDCR_EL2.E2TB](#) is 0b00, this field reads as one from EL1.
- Otherwise, this field reads as zero.

Align, bits [3:0]

Defines the minimum alignment constraint for writes to [TRBPTR_EL1](#) and [TRBTRG_EL1](#). Defined values are:

| Align | Meaning |
|--------|-------------|
| 0b0000 | Byte. |
| 0b0001 | Halfword. |
| 0b0010 | Word. |
| 0b0011 | Doubleword. |
| 0b0100 | 16 bytes. |
| 0b0101 | 32 bytes. |
| 0b0110 | 64 bytes. |
| 0b0111 | 128 bytes. |
| 0b1000 | 256 bytes. |
| 0b1001 | 512 bytes. |
| 0b1010 | 1KB. |
| 0b1011 | 2KB. |

All other values are reserved.

Access to this field is **RO**.

Accessing TRBIDR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRBIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRBIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        return TRBIDR_EL1;
elsif PSTATE.EL == EL2 then
    return TRBIDR_EL1;
elsif PSTATE.EL == EL3 then
    return TRBIDR_EL1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBLIMITR_EL1, Trace Buffer Limit Address Register

The TRBLIMITR_EL1 characteristics are:

Purpose

Defines the top address for the trace buffer, and controls the trace buffer modes and enable.

Configuration

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBLIMITR_EL1 are UNDEFINED.

Attributes

TRBLIMITR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|-------|----|----|----|----|----|-----|----|----|----|----|----|----|----|----|----|--|--|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | |
| | | | | | | | | | | | | | | | | LIMIT | | | | | | | | | | | | | | | | | | | | | |
| LIMIT | | | | | | | | | | | | RES0 | | | | | | | | | | nVM | TM | FM | E | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | |

LIMIT, bits [63:12]

Trace Buffer Limit pointer address. (TRBLIMITR_EL1.LIMIT << 12) is the address of the last byte in the trace buffer plus one. Bits [11:0] of the Limit pointer address are always zero. If the smallest implemented translation granule is not 4KB, then TRBLIMITR_EL1[N-1:12] are RES0, where N is the IMPLEMENTATION DEFINED value $\text{Log}_2(\text{smallest implemented translation granule})$.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [11:6]

Reserved, RES0.

nVM, bit [5]

Address mode.

| nVM | Meaning |
|-----|--|
| 0b0 | The trace buffer pointers are virtual addresses. |
| 0b1 | The trace buffer pointers are: <ul style="list-style-type: none"> Physical address in the owning security state if the owning translation regime has no stage 2 translation. Intermediate physical addresses in the owning security state if the owning translation regime has stage 2 translations. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

TM, bits [4:3]

Trigger mode.

| TM | Meaning |
|-----------|---|
| 0b00 | Stop on trigger. Flush then stop collection and raise maintenance interrupt on Trigger Event. |
| 0b01 | IRQ on trigger. Continue collection and raise maintenance interrupt on Trigger Event. |
| 0b11 | Ignore trigger. Continue collection and do not raise maintenance interrupt on Trigger Event. |

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

FM, bits [2:1]

Trace buffer mode.

| FM | Meaning |
|-----------|---|
| 0b00 | Fill mode. Stop collection and raise maintenance interrupt on current write pointer wrap. |
| 0b01 | Wrap mode. Continue collection and raise maintenance interrupt on current write pointer wrap. |
| 0b11 | Circular Buffer mode. Continue collection and do not raise maintenance interrupt on current write pointer wrap. |

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Trace Buffer Unit enable.

| E | Meaning |
|----------|--|
| 0b0 | Trace Buffer Unit disabled. |
| 0b1 | Trace Buffer Unit enabled by this control. |

Regardless of the value of this field, the Trace Buffer Unit is disabled when SelfHostedTraceEnabled() == FALSE. All output is discarded by the Trace Buffer Unit when it is disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing TRBLIMITR_EL1

The PE might ignore a direct write to TRBLIMITR_EL1, other than a direct write that modifies TRBLIMITR_EL1.E, if TRBLIMITR_EL1.E == 1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRBLIMITR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRBLIMITR_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRBLIMITR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRBLIMITR_EL1;
elsif PSTATE.EL == EL3 then
    return TRBLIMITR_EL1;

```

MSR TRBLIMITR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRBLIMITR_EL1 ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBLIMITR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBLIMITR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TRBLIMITR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBMAR_EL1, Trace Buffer Memory Attribute Register

The TRBMAR_EL1 characteristics are:

Purpose

Controls Trace Buffer Unit accesses to memory.

If the trace buffer pointers specify virtual addresses, the address properties are defined by the translation tables and this register is ignored.

Configuration

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBMAR_EL1 are UNDEFINED.

Attributes

TRBMAR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|-----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | SH | | | | Attr[7:4] | | | | Attr[3:0] | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:10]

Reserved, RES0.

SH, bits [9:8]

Trace buffer shareability domain. Defines the shareability domain for Normal memory used by the trace buffer.

| SH | Meaning |
|------|------------------|
| 0b00 | Non-shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

All other values are reserved.

This field is ignored when TRBMAR_EL1.Attr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Attr[7:4], bits [7:4]
When TRBMAR_EL1.Attr[3:0] == 0b0000:

Trace buffer memory type and attributes. Defines the memory type and, for Normal memory, the cacheability attributes, for memory addressed by the trace buffer.

| Attr[7:4] | Meaning | Applies when |
|-----------|---|-------------------------------|
| 0b0000 | Device-nGnRnE memory. | |
| 0b0100 | Normal memory, Inner Non-cacheable, Outer Non-cacheable with the XS attribute set to 0. | When FEAT_XS is implemented |
| 0b1010 | Normal memory, Inner Write-through Cacheable, Outer Write-through Cacheable, Non-transient, Read-Allocate with the XS attribute set to 0. | When FEAT_XS is implemented |
| 0b1111 | Tagged Normal memory, Outer Write-Back Non-transient, Read-allocate Write-allocate. | When FEAT_MTE2 is implemented |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Trace buffer memory type and attributes. Defines the memory type and, for Normal memory, the Outer cacheability attributes, for memory addressed by the trace buffer.

| Attr[7:4] | Meaning |
|-----------|---|
| 0b0000 | Device memory. The Device memory type is defined by TRBMAR_EL1.Attr[3:0]. |
| 0b0001 | Normal memory, Outer Write-Through Transient, Write-allocate. |
| 0b0010 | Normal memory, Outer Write-Through Transient, Read-allocate. |
| 0b0011 | Normal memory, Outer Write-Through Transient, Read-allocate Write-allocate. |
| 0b0100 | Normal memory, Outer Non-cacheable. |
| 0b0101 | Normal memory, Outer Write-Back Transient, Write-allocate. |
| 0b0110 | Normal memory, Outer Write-Back Transient, Read-allocate. |
| 0b0111 | Normal memory, Outer Write-Back Transient, Read-allocate Write-allocate. |
| 0b1000 | Normal memory, Outer Write-Through Non-transient, No allocate. |
| 0b1001 | Normal memory, Outer Write-Through Non-transient, Write-allocate. |
| 0b1010 | Normal memory, Outer Write-Through Non-transient, Read-allocate. |
| 0b1011 | Normal memory, Outer Write-Through Non-transient, Read-allocate Write-allocate. |
| 0b1100 | Normal memory, Outer Write-Back Non-transient, No allocate. |
| 0b1101 | Normal memory, Outer Write-Back Non-transient, Write-allocate. |
| 0b1110 | Normal memory, Outer Write-Back Non-transient, Read-allocate. |
| 0b1111 | Normal memory, Outer Write-Back Non-transient, Read-allocate Write-allocate. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Attr[3:0], bits [3:0]**When TRBMAR_EL1.Attr[7:4] == 0b0000:**

Trace buffer memory attributes. Defines the Device memory attributes for memory addressed by the trace buffer.

| Attr[3:0] | Meaning | Applies when |
|-----------|--|-----------------------------|
| 0b0000 | Device-nGnRnE memory. | |
| 0b0100 | Device-nGnRE memory. | |
| 0b1000 | Device-nGRE memory. | |
| 0b1100 | Device-GRE memory. | |
| 0b0001 | Device-nGnRnE memory with the XS attribute set to 0. | When FEAT_XS is implemented |
| 0b0101 | Device-nGnRE memory with the XS attribute set to 0. | When FEAT_XS is implemented |
| 0b1001 | Device-nGRE memory with the XS attribute set to 0. | When FEAT_XS is implemented |
| 0b1101 | Device-GRE memory with the XS attribute set to 0. | When FEAT_XS is implemented |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Trace buffer memory attributes. Defines the Inner cacheability attributes for memory addressed by the trace buffer.

| Attr[3:0] | Meaning | Applies when |
|-----------|---|--|
| 0b0000 | The memory type is defined by TRBMAR_EL1.Attr[7:4]. | When FEAT_MTE is implemented or FEAT_XS is implemented |
| 0b0001 | Normal memory, Inner Write-Through Transient, Write-allocate. | |
| 0b0010 | Normal memory, Inner Write-Through Transient, Read-allocate. | |
| 0b0011 | Normal memory, Inner Write-Through Transient, Read-allocate Write-allocate. | |
| 0b0100 | Normal memory, Inner Non-cacheable. | |
| 0b0101 | Normal memory, Inner Write-Back Transient, Write-allocate. | |
| 0b0110 | Normal memory, Inner Write-Back Transient, Read-allocate. | |
| 0b0111 | Normal memory, Inner Write-Back Transient, Read-allocate Write-allocate. | |
| 0b1000 | Normal memory, Inner Write-Through Non-transient, No allocate. | |
| 0b1001 | Normal memory, Inner Write-Through Non-transient, Write-allocate. | |
| 0b1010 | Normal memory, Inner Write-Through Non-transient, Read-allocate. | |
| 0b1011 | Normal memory, Inner Write-Through Non-transient, Read-allocate Write-allocate. | |
| 0b1100 | Normal memory, Inner Write-Back Non-transient, No allocate. | |
| 0b1101 | Normal memory, Inner Write-Back Non-transient, Write-allocate. | |
| 0b1110 | Normal memory, Inner Write-Back Non-transient, Read-allocate. | |
| 0b1111 | Normal memory, Inner Write-Back Non-transient, Read-allocate Write-allocate. | |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TRBMAR_EL1

The PE might ignore a direct write to TRBMAR_EL1 if [TRBLIMITR_EL1.E](#) == 1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRBMAR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRBMAR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRBMAR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRBMAR_EL1;
elsif PSTATE.EL == EL3 then
    return TRBMAR_EL1;
    
```

MSR TRBMAR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRBMAR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBMAR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBMAR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TRBMAR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBPTR_EL1, Trace Buffer Write Pointer Register

The TRBPTR_EL1 characteristics are:

Purpose

Defines the current write pointer for the trace buffer.

Configuration

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBPTR_EL1 are UNDEFINED.

Attributes

TRBPTR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | PTR | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | PTR | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

PTR, bits [63:0]

Trace Buffer current write pointer address.

Defines the virtual address of the next entry to be written to the trace buffer.

The architecture places restrictions on the values that software can write to the pointer.

Note

As a result of the restrictions an implementation might treat some of PTR[M:0] as RES0, where M is defined by [TRBIDR_EL1](#).Align.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing TRBPTR_EL1

The PE might ignore a direct write to TRBPTR_EL1 if [TRBLIMITR_EL1](#).E == 1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRBPTR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRBPTR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRBPTR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRBPTR_EL1;
elsif PSTATE.EL == EL3 then
    return TRBPTR_EL1;

```

MSR TRBPTR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRBPTR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBPTR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBPTR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TRBPTR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBSR_EL1, Trace Buffer Status/syndrome Register

The TRBSR_EL1 characteristics are:

Purpose

Provides syndrome information to software for a trace buffer management event.

Configuration

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBSR_EL1 are UNDEFINED.

Attributes

TRBSR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|------|----|----|----|-----|-----|------|------|----|----|------|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| EC | | | | | | RES0 | | | | IRQ | TRG | WRAP | RES0 | EA | S | RES0 | MSS | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

EC, bits [31:26]

Event class. Top-level description of the cause of the trace buffer management event.

| EC | Meaning | MSS | Applies when |
|----------|--|--|------------------------------|
| 0b000000 | Other trace buffer management event. All trace buffer management events other than those described by the other defined Event class codes. | MSS encoding for other trace buffer management events | |
| 0b011110 | Granule Protection Check fault, other than GPF, on write to trace buffer. | MSS encoding for Granule Protection Check fault | When FEAT_RME is implemented |
| 0b011111 | Buffer management event for IMPLEMENTATION DEFINED reason. | MSS encoding for Buffer management event for IMPLEMENTATION DEFINED reason | |
| 0b100100 | Stage 1 Data Abort on write to trace buffer. | MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer | |
| 0b100101 | Stage 2 Data Abort on write to trace buffer. | MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer | |

All other values are reserved.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [25:23]

Reserved, RES0.

IRQ, bit [22]

Maintenance interrupt status.

| IRQ | Meaning |
|-----|--|
| 0b0 | Maintenance interrupt is not asserted. |
| 0b1 | Maintenance interrupt is asserted. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

TRG, bit [21]

Triggered.

| TRG | Meaning |
|-----|--|
| 0b0 | No Detected Trigger has been observed since this field was last cleared to zero. |
| 0b1 | A Detected Trigger has been observed since this field was last cleared to zero. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

WRAP, bit [20]

Wrapped.

| WRAP | Meaning |
|-------------|--|
| 0b0 | The current write pointer has not wrapped since this field was last cleared to zero. |
| 0b1 | The current write pointer has wrapped since this field was last cleared to zero. |

For each byte of trace the Trace Buffer Unit Accepts and writes to the trace buffer at the address in the current write pointer, if the current write pointer is equal to the Limit pointer minus one, the current write pointer is wrapped by setting it to the Base pointer, and this field is set to 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [19]

Reserved, RES0.

EA, bit [18]

External Abort.

| EA | Meaning |
|-----------|--|
| 0b0 | An External Abort has not been asserted. |
| 0b1 | An External Abort has been asserted and detected by the Trace Buffer Unit. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When the PE never sets this field as the result of an External Abort, access to this field is **RES0**.
- Otherwise, access to this field is **RW**.

S, bit [17]

Stopped.

| S | Meaning |
|----------|----------------------------------|
| 0b0 | Collection has not been stopped. |
| 0b1 | Collection is stopped. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bit [16]

Reserved, RES0.

MSS, bits [15:0]

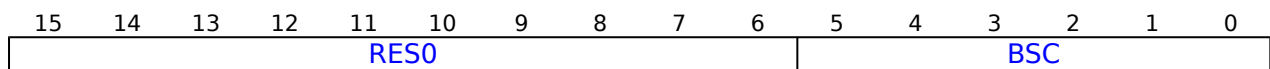
Management Event Specific Syndrome. Contains syndrome specific to the management event.

The syndrome contents for each management event are described in the following sections.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

MSS encoding for other trace buffer management events



Bits [15:6]

Reserved, RES0.

BSC, bits [5:0]

Trace buffer status code.

| BSC | Meaning |
|----------|---|
| 0b000000 | Collection not stopped. |
| 0b000001 | Trace buffer filled. Collection stopped because the current write pointer wrapped to the base pointer and the trace buffer mode is Fill mode. |
| 0b000010 | Trigger Event. Collection stopped because of a Trigger Event. See TRBTRG_EL1 for more information. |

All other values are reserved.

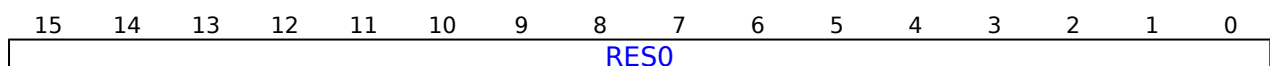
MSS encoding for Buffer management event for IMPLEMENTATION DEFINED reason



IMPLEMENTATION DEFINED, bits [15:0]

IMPLEMENTATION DEFINED.

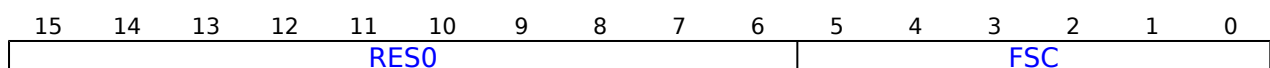
MSS encoding for Granule Protection Check fault



Bits [15:0]

Reserved, RES0.

MSS encoding for stage 1 or stage 2 Data Aborts on write to trace buffer



Bits [15:6]

Reserved, RES0.

FSC, bits [5:0]

Fault status code.

| FSC | Meaning | Applies when |
|------------|---|--|
| 0b000000 | Address size fault, level 0 of translation or translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000100 | Translation fault, level 0. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001000 | Access flag fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001100 | Permission fault, level 0. | When FEAT_LPA2 is implemented |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010000 | Synchronous External abort, not on translation table walk or hardware update of translation table. | |
| 0b010001 | Asynchronous External abort. | |
| 0b010011 | Synchronous External abort on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented |
| 0b010100 | Synchronous External abort on translation table walk or hardware update of translation table, level 0. | |
| 0b010101 | Synchronous External abort on translation table walk or hardware update of translation table, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk or hardware update of translation table, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk or hardware update of translation table, level 3. | |
| 0b011011 | Synchronous parity or ECC error on memory access on translation table walk or hardware update of translation table, level -1. | When FEAT_LPA2 is implemented and FEAT_RAS is not implemented |
| 0b100001 | Alignment fault. | |
| 0b100011 | Granule Protection Fault on translation table walk or hardware update of translation table, level -1. | When FEAT_RME is implemented and FEAT_LPA2 is implemented |
| 0b100100 | Granule Protection Fault on translation table walk or hardware update of translation table, level 0. | When FEAT_RME is implemented |
| 0b100101 | Granule Protection Fault on translation table walk or hardware update of translation table, level 1. | When FEAT_RME is implemented |

| | | |
|----------|--|---------------------------------|
| 0b100110 | Granule Protection Fault on translation table walk or hardware update of translation table, level 2. | When FEAT_RME is implemented |
| 0b100111 | Granule Protection Fault on translation table walk or hardware update of translation table, level 3. | When FEAT_RME is implemented |
| 0b101000 | Granule Protection Fault, not on translation table walk or hardware update of translation table. | When FEAT_RME is implemented |
| 0b101001 | Address size fault, level -1. | When FEAT_LPA2 is implemented |
| 0b101011 | Translation fault, level -1. | When FEAT_LPA2 is implemented |
| 0b110000 | TLB conflict abort. | When FEAT_HAFDBS is implemented |
| 0b110001 | Unsupported atomic hardware update fault. | |

All other values are reserved.

Accessing TRBSR_EL1

The PE might ignore a direct write to TRBSR_EL1 if [TRBLIMITR_EL1.E](#) == 1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRBSR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRBSR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRBSR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRBSR_EL1;
elsif PSTATE.EL == EL3 then
    return TRBSR_EL1;

```

MSR TRBSR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.TRBSR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBSR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBSR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TRBSR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRBTRG_EL1, Trace Buffer Trigger Counter Register

The TRBTRG_EL1 characteristics are:

Purpose

Specifies the number of bytes of trace to capture following a Detected Trigger before a Trigger Event.

Configuration

This register is present only when FEAT_TRBE is implemented. Otherwise, direct accesses to TRBTRG_EL1 are UNDEFINED.

Attributes

TRBTRG_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | TRG | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

TRG, bits [31:0]

Trigger count.

Specifies the number of bytes of trace to capture following a Detected Trigger before a Trigger Event.

TRBTRG_EL1 decrements by 1 for every byte of trace written to the trace buffer when all of the following are true:

- TRBTRG_EL1 is nonzero.
- [TRBSR_EL1](#).TRG is 1.

The architecture places restrictions on the values that software can write to the counter.

Note

As a result of the restrictions an implementation might treat some of TRG[M:0] as RES0, where M is defined by [TRBIDR_EL1](#).Align.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing TRBTRG_EL1

The PE might ignore a direct write to TRBTRG_EL1 if [TRBLIMITR_EL1](#).E == 1.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRBTRG_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRBTRG_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRBTRG_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRBTRG_EL1;
elsif PSTATE.EL == EL3 then
    return TRBTRG_EL1;
    
```

MSR TRBTRG_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.TRBTRG_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.E2TB == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBTRG_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        UNDEFINED;
    elsif HaveEL(EL3) && (MDCR_EL3.NSTB[0] == '0' || MDCR_EL3.NSTB[1] != SCR_EL3.NS ||
(IsFeatureImplemented(FEAT_RME) && MDCR_EL3.NSTBE != SCR_EL3.NSE)) then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRBTRG_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TRBTRG_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCACATR<n>, Address Comparator Access Type Register <n>, n = 0 - 15

The TRCACATR<n> characteristics are:

Purpose

Defines the type of access for the corresponding [TRCACVR<n>](#) Register. This register configures the context type, Exception levels, alignment, masking that is applied by the Address Comparator, and how the Address Comparator behaves when it is one half of an Address Range Comparator.

Configuration

AArch64 System register TRCACATR<n> bits [63:0] are architecturally mapped to External register [TRCACATR<n>\[63:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMACPAIRS * 2 > n. Otherwise, direct accesses to TRCACATR<n> are UNDEFINED.

Attributes

TRCACATR<n> is a 64-bit register.

Field descriptions

| | | | | | | |
|----------------------------|----|----|----|----|----|----|
| 63626160595857565554535251 | 50 | 49 | 48 | 47 | 46 | 45 |
| | | | | | | |
| | | | | | | |
| 31302928272625242322212019 | 18 | 17 | 16 | 15 | 14 | 13 |

Bits [63:19]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [18]

When TRCIDR6.EXLEVEL_RL_EL2 == 1:

Realm EL2 address comparison control. Controls whether a comparison can occur at EL2 in Realm state.

| EXLEVEL_RL_EL2 | Meaning |
|----------------|--|
| 0b0 | When TRCACATR<n>.EXLEVEL_NS_EL2 is 0 the Address Comparator performs comparisons in Realm EL2. When TRCACATR<n>.EXLEVEL_NS_EL2 is 1 the Address Comparator does not perform comparisons in Realm EL2. |
| 0b1 | When TRCACATR<n>.EXLEVEL_NS_EL2 is 0 the Address Comparator does not perform comparisons in Realm EL2. When TRCACATR<n>.EXLEVEL_NS_EL2 is 1 the Address Comparator performs comparisons in Realm EL2. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL1, bit [17]**When TRCIDR6.EXLEVEL_RL_EL1 == 1:**

Realm EL1 address comparison control. Controls whether a comparison can occur at EL1 in Realm state.

| EXLEVEL_RL_EL1 | Meaning |
|----------------|--|
| 0b0 | When TRCACATR<n>.EXLEVEL_NS_EL1 is 0 the Address Comparator performs comparisons in Realm EL1. When TRCACATR<n>.EXLEVEL_NS_EL1 is 1 the Address Comparator does not perform comparisons in Realm EL1. |
| 0b1 | When TRCACATR<n>.EXLEVEL_NS_EL1 is 0 the Address Comparator does not perform comparisons in Realm EL1. When TRCACATR<n>.EXLEVEL_NS_EL1 is 1 the Address Comparator performs comparisons in Realm EL1. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL0, bit [16]**When TRCIDR6.EXLEVEL_RL_EL0 == 1:**

Realm EL0 address comparison control. Controls whether a comparison can occur at EL0 in Realm state.

| EXLEVEL_RL_EL0 | Meaning |
|----------------|--|
| 0b0 | When TRCACATR<n>.EXLEVEL_NS_EL0 is 0 the Address Comparator performs comparisons in Realm EL0. When TRCACATR<n>.EXLEVEL_NS_EL0 is 1 the Address Comparator does not perform comparisons in Realm EL0. |
| 0b1 | When TRCACATR<n>.EXLEVEL_NS_EL0 is 0 the Address Comparator does not perform comparisons in Realm EL0. When TRCACATR<n>.EXLEVEL_NS_EL0 is 1 the Address Comparator performs comparisons in Realm EL0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [15]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [14]**When Non-secure EL2 is implemented:**

Non-secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Non-secure state.

| EXLEVEL_NS_EL2 | Meaning |
|----------------|--|
| 0b0 | The Address Comparator performs comparisons in Non-secure EL2. |
| 0b1 | The Address Comparator does not perform comparisons in Non-secure EL2. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL1, bit [13]**When Non-secure EL1 is implemented:**

Non-secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Non-secure state.

| EXLEVEL_NS_EL1 | Meaning |
|----------------|--|
| 0b0 | The Address Comparator performs comparisons in Non-secure EL1. |
| 0b1 | The Address Comparator does not perform comparisons in Non-secure EL1. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL0, bit [12]**When Non-secure EL0 is implemented:**

Non-secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Non-secure state.

| EXLEVEL_NS_EL0 | Meaning |
|----------------|--|
| 0b0 | The Address Comparator performs comparisons in Non-secure EL0. |
| 0b1 | The Address Comparator does not perform comparisons in Non-secure EL0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL3, bit [11]**When EL3 is implemented:**

EL3 address comparison control. Controls whether a comparison can occur at EL3.

| EXLEVEL_S_EL3 | Meaning |
|----------------------|---|
| 0b0 | The Address Comparator performs comparisons in EL3. |
| 0b1 | The Address Comparator does not perform comparisons in EL3. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL2, bit [10]**When EL2 is implemented and FEAT_SEL2 is implemented:**

Secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Secure state.

| EXLEVEL_S_EL2 | Meaning |
|----------------------|--|
| 0b0 | The Address Comparator performs comparisons in Secure EL2. |
| 0b1 | The Address Comparator does not perform comparisons in Secure EL2. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL1, bit [9]**When Secure EL1 is implemented:**

Secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Secure state.

| EXLEVEL_S_EL1 | Meaning |
|----------------------|--|
| 0b0 | The Address Comparator performs comparisons in Secure EL1. |
| 0b1 | The Address Comparator does not perform comparisons in Secure EL1. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL0, bit [8]**When Secure EL0 is implemented:**

Secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Secure state.

| EXLEVEL S EL0 | Meaning |
|---------------|--|
| 0b0 | The Address Comparator performs comparisons in Secure EL0. |
| 0b1 | The Address Comparator does not perform comparisons in Secure EL0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

CONTEXT, bits [6:4]

When TRCIDR4.NUMCIDC != 0b0000 or TRCIDR4.NUMVMIDC != 0b0000:

Selects a Context Identifier Comparator or Virtual Context Identifier Comparator:

| CONTEXT | Meaning | Applies when |
|---------|---------------|--|
| 0b000 | Comparator 0. | |
| 0b001 | Comparator 1. | When TRCIDR4.NUMCIDC > 0b0001 or TRCIDR4.NUMVMIDC > 0b0001 |
| 0b010 | Comparator 2. | When TRCIDR4.NUMCIDC > 0b0010 or TRCIDR4.NUMVMIDC > 0b0010 |
| 0b011 | Comparator 3. | When TRCIDR4.NUMCIDC > 0b0011 or TRCIDR4.NUMVMIDC > 0b0011 |
| 0b100 | Comparator 4. | When TRCIDR4.NUMCIDC > 0b0100 or TRCIDR4.NUMVMIDC > 0b0100 |
| 0b101 | Comparator 5. | When TRCIDR4.NUMCIDC > 0b0101 or TRCIDR4.NUMVMIDC > 0b0101 |
| 0b110 | Comparator 6. | When TRCIDR4.NUMCIDC > 0b0110 or TRCIDR4.NUMVMIDC > 0b0110 |
| 0b111 | Comparator 7. | When TRCIDR4.NUMCIDC > 0b0111 or TRCIDR4.NUMVMIDC > 0b0111 |

The width of this field is dependent on the maximum number of Context Identifier Comparators or Virtual Context Identifier Comparators implemented. Unimplemented bits are RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CONTEXTTYPE, bits [3:2]

When TRCIDR4.NUMCIDC != 0b0000 or TRCIDR4.NUMVMIDC != 0b0000:

Controls whether the Address Comparator is dependent on a Context Identifier Comparator, a Virtual Context Identifier Comparator, or both comparisons.

| CONTEXTTYPE | Meaning | Applies when |
|-------------|---|---|
| 0b00 | The Address Comparator is not dependent on the Context Identifier Comparators or Virtual Context Identifier Comparators. | |
| 0b01 | The Address Comparator is dependent on the Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if both the Context Identifier Comparator and the address comparison match. | When TRCIDR4.NUMCIDC != 0b0000 |
| 0b10 | The Address Comparator is dependent on the Virtual Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if both the Virtual Context Identifier Comparator and the address comparison match. | When TRCIDR4.NUMVMIDC != 0b0000 |
| 0b11 | The Address Comparator is dependent on the Context Identifier Comparator and Virtual Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if the Context Identifier Comparator, the Virtual Context Identifier Comparator, and address comparison all match. | When TRCIDR4.NUMCIDC != 0b0000 and TRCIDR4.NUMVMIDC != 0b0000 |

If [TRCIDR4.NUMCIDC](#) == 0b0000, then bit [2] is RES0.

If [TRCIDR4.NUMVMIDC](#) == 0b0000, then bit [3] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [1:0]

Reserved, RES0.

Accessing TRCACATR<n>

Must be programmed if any of the following are true:

- [TRCBBCTLR](#).RANGE[n/2] == 1.
- [TRCRSCTLR<a>](#).GROUP == 0b0100 and [TRCRSCTLR<a>](#).SAC[n] == 1.
- [TRCRSCTLR<a>](#).GROUP == 0b0101 and [TRCRSCTLR<a>](#).ARC[n/2] == 1.
- [TRCVIIECTLR](#).EXCLUDE[n/2] == 1.
- [TRCVIIECTLR](#).INCLUDE[n/2] == 1.
- [TRCVISSCTLR](#).START[n] == 1.
- [TRCVISSCTLR](#).STOP[n] == 1.
- TRCSSCCR<>.ARC[n/2] == 1.
- TRCSSCCR<>.SAC[n] == 1.
- [TRCQCTLR](#).RANGE[n/2] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCACATR<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-----------|
| 0b10 | 0b001 | 0b0010 | n[2:0]:0b0 | 0b01:n[3] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCACATR[UInt(op2<0>:CRm<3:1>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCACATR[UInt(op2<0>:CRm<3:1>)];
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCACATR[UInt(op2<0>:CRm<3:1>)];

```

MSR TRCACATR<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-----------|
| 0b10 | 0b001 | 0b0010 | n[2:0]:0b0 | 0b01:n[3] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCACATR[UInt(op2<0>:CRm<3:1>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCACATR[UInt(op2<0>:CRm<3:1>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCACATR[UInt(op2<0>:CRm<3:1>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCACVR<n>, Address Comparator Value Register <n>, n = 0 - 15

The TRCACVR<n> characteristics are:

Purpose

Contains the address value.

Configuration

AArch64 System register TRCACVR<n> bits [63:0] are architecturally mapped to External register [TRCACVR<n>\[63:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMACPAIRS * 2 > n. Otherwise, direct accesses to TRCACVR<n> are UNDEFINED.

Attributes

TRCACVR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| | | | | | | | | | | | | | | | | ADDRESS | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ADDRESS | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

ADDRESS, bits [63:0]

Address Value.

The Address Comparators can support implementations that use multiple address widths. When the trace unit compares the ADDRESS field with an address that has a width less than this field, then the address must be zero-extended to the ADDRESS field width. The trace unit then compares all implemented bits. For example, in a system that supports both 32-bit and 64-bit addresses, when the PE is in AArch32 state the comparator must zero-extend the 32-bit address and compare against the full 64 bits that are stored in the TRCACVR<n>. This requires that the trace analyzer always programs all implemented bits of the TRCACVR<n>.

The result of writing a value other than all zeros or all ones to ADDRESS at bits[63:P] is an UNKNOWN value, where P is defined as the maximum virtual address size supported by the PE.

The result of writing a value of all zeros or all ones to ADDRESS at bits[63:P] is the written value, and a read of the register returns the written value.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCACVR<n>

Must be programmed if any of the following are true:

- [TRCBBCTLR](#).RANGE[n/2] == 1.
- [TRCRSCTLR<a>](#).GROUP == 0b0100 and [TRCRSCTLR<a>](#).SAC[n] == 1.
- [TRCRSCTLR<a>](#).GROUP == 0b0101 and [TRCRSCTLR<a>](#).ARC[n/2] == 1.
- [TRCVIIETLR](#).EXCLUDE[n/2] == 1.
- [TRCVIIETLR](#).INCLUDE[n/2] == 1.

- [TRCVISSCTLR](#).START[n] == 1.
- [TRCVISSCTLR](#).STOP[n] == 1.
- TRCSSCCR<>.ARC[n/2] == 1.
- TRCSSCCR<>.SAC[n] == 1.
- [TRCQCTL](#)R.RANGE[n/2] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCACVR<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-----------|
| 0b10 | 0b001 | 0b0010 | n[2:0]:0b0 | 0b00:n[3] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCACVR[UInt(op2<0>:CRm<3:1>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCACVR[UInt(op2<0>:CRm<3:1>)];
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCACVR[UInt(op2<0>:CRm<3:1>)];

```

MSR TRCACVR<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-----------|
| 0b10 | 0b001 | 0b0010 | n[2:0]:0b0 | 0b00:n[3] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCACVR[UInt(op2<0>:CRm<3:1>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCACVR[UInt(op2<0>:CRm<3:1>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCACVR[UInt(op2<0>:CRm<3:1>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCAUTHSTATUS, Authentication Status Register

The TRCAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

For additional information, see the CoreSight Architecture Specification.

Configuration

AArch64 System register TRCAUTHSTATUS bits [31:0] are architecturally mapped to External register [TRCAUTHSTATUS\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCAUTHSTATUS are UNDEFINED.

Attributes

TRCAUTHSTATUS is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|----|----|------|----|----|----|------|----|----|----|------|----|----|----|----|----|----|----|-------|----|----|----|------|----|----|----|------|--|--|--|-----|--|--|--|------|--|--|--|-----|--|--|--|-------|--|--|--|------|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | RTNID | | | | RTID | | | | RES0 | | | | | | | | | | | | RLNID | | | | RLID | | | | HNID | | | | HID | | | | SNID | | | | SID | | | | NSNID | | | | NSID | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:28]

Reserved, RES0.

RTNID, bits [27:26]

Root non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RTNID.

RTID, bits [25:24]

Root invasive debug.

| RTID | Meaning |
|------|------------------|
| 0b00 | Not implemented. |

Bits [23:16]

Reserved, RES0.

RLNID, bits [15:14]

Realm non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RLNID.

RLID, bits [13:12]

Realm invasive debug.

| RLID | Meaning |
|------|------------------|
| 0b00 | Not implemented. |

HNID, bits [11:10]

Hyp Non-invasive Debug. Indicates whether a separate enable control for EL2 non-invasive debug features is implemented and enabled.

| HNID | Meaning |
|------|---|
| 0b00 | Separate Hyp non-invasive debug enable not implemented, or EL2 non-invasive debug features not implemented. |
| 0b10 | Implemented and disabled. |
| 0b11 | Implemented and enabled. |

All other values are reserved.

This field reads as 0b00.

HID, bits [9:8]

Hyp Invasive Debug. Indicates whether a separate enable control for EL2 invasive debug features is implemented and enabled.

| HID | Meaning |
|------|---|
| 0b00 | Separate Hyp invasive debug enable not implemented, or EL2 invasive debug features not implemented. |
| 0b10 | Implemented and disabled. |
| 0b11 | Implemented and enabled. |

All other values are reserved.

This field reads as 0b00.

SNID, bits [7:6]

Secure Non-invasive Debug. Indicates whether Secure non-invasive debug features are implemented and enabled.

| SNID | Meaning |
|------|---|
| 0b00 | Secure non-invasive debug features not implemented. |
| 0b10 | Implemented and disabled. |
| 0b11 | Implemented and enabled. |

All other values are reserved.

When EL3 is implemented, this field takes the value 0b10 or 0b11 depending whether Secure non-invasive debug is enabled.

When EL3 is not implemented and the PE is Non-secure, this field reads as 0b00.

When EL3 is not implemented and the PE is Secure, this field takes the value 0b10 or 0b11 depending whether Secure non-invasive debug is enabled.

SID, bits [5:4]

Secure Invasive Debug. Indicates whether Secure invasive debug features are implemented and enabled.

| SID | Meaning |
|------|---|
| 0b00 | Secure invasive debug features not implemented. |
| 0b10 | Implemented and disabled. |
| 0b11 | Implemented and enabled. |

All other values are reserved.

This field reads as 0b00.

NSNID, bits [3:2]

Non-secure Non-invasive Debug. Indicates whether Non-secure non-invasive debug features are implemented and enabled.

| NSNID | Meaning |
|-------|---|
| 0b00 | Non-secure non-invasive debug features not implemented. |
| 0b10 | Implemented and disabled. |
| 0b11 | Implemented and enabled. |

All other values are reserved.

When EL3 is implemented, this field reads as 0b11.

When EL3 is not implemented and the PE is Non-secure, this field reads as 0b11.

When EL3 is not implemented and the PE is Secure, this field reads as 0b00.

NSID, bits [1:0]

Non-secure Invasive Debug. Indicates whether Non-secure invasive debug features are implemented and enabled.

| NSID | Meaning |
|------|---|
| 0b00 | Non-secure invasive debug features not implemented. |
| 0b10 | Implemented and disabled. |
| 0b11 | Implemented and enabled. |

All other values are reserved.

This field reads as 0b00.

Accessing TRCAUTHSTATUS

For implementations that support multiple access mechanisms, different access mechanisms can return different values for reads of TRCAUTHSTATUS if the authentication signals have changed and that change has not yet been synchronized by a Context synchronization event. This scenario can happen if, for example, the external debugger view is implemented separately from the system instruction view to allow for separate power domains, and so observes changes on the signals differently.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCAUTHSTATUS

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0111 | 0b1110 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCAUTHSTATUS ==
'1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCAUTHSTATUS;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCAUTHSTATUS;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCAUTHSTATUS;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCAUXCTLR, Auxiliary Control Register

The TRCAUXCTLR characteristics are:

Purpose

The function of this register is IMPLEMENTATION DEFINED.

Configuration

AArch64 System register TRCAUXCTLR bits [31:0] are architecturally mapped to External register [TRCAUXCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCAUXCTLR are UNDEFINED.

Attributes

TRCAUXCTLR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to 0.

Accessing TRCAUXCTLR

If this register is nonzero then it might cause the behavior of a trace unit to contradict this architecture specification. See the documentation of the specific implementation for information about the IMPLEMENTATION DEFINED support for this register.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCAUXCTLR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCAUXCTLR == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCAUXCTLR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCAUXCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCAUXCTLR;

```

MSR TRCAUXCTLR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRCAUXCTLR == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCAUXCTLR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCAUXCTLR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCAUXCTLR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCBBCTLR, Branch Broadcast Control Register

The TRCBBCTLR characteristics are:

Purpose

Controls the regions in the memory map where branch broadcasting is active.

Configuration

AArch64 System register TRCBBCTLR bits [31:0] are architecturally mapped to External register [TRCBBCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented, TRCIDR0.TRCBB == 1 and TRCIDR4.NUMACPAIRS > 0b0000. Otherwise, direct accesses to TRCBBCTLR are UNDEFINED.

Attributes

TRCBBCTLR is a 64-bit register.

Field descriptions

| | | | | | | | | | | |
|--|----|----|----|------|----------|----------|----------|----------|----------|----------|
| 6362616059585756555453525150494847464544434241 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | | | |
| RES0 | | | | | | | | | | |
| RES0 | | | | MODE | RANGE[7] | RANGE[6] | RANGE[5] | RANGE[4] | RANGE[3] | RANGE[2] |
| 31302928272625242322212019181716151413121110 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | | |

Bits [63:9]

Reserved, RES0.

MODE, bit [8]

Mode.

| MODE | Meaning |
|------|---|
| 0b0 | Exclude Mode. Branch broadcasting is not active for instructions in the address ranges defined by TRCBBCTLR.RANGE. If TRCBBCTLR.RANGE == 0x00 then branch broadcasting is active for all instructions. |
| 0b1 | Include Mode. Branch broadcasting is active for instructions in the address ranges defined by TRCBBCTLR.RANGE. If TRCBBCTLR.RANGE == 0x00 then the behavior of the trace unit is CONSTRAINED UNPREDICTABLE. That is, the trace unit might or might not consider any instructions to be in a branch broadcasting region. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RANGE[<m>], bit [m], for m = 7 to 0

Address range field.

Selects whether Address Range Comparator <m> is used with branch broadcasting.

| RANGE[<m>] | Meaning |
|------------|---|
| 0b0 | The address range that Address Range Comparator <m> defines, is not selected. |
| 0b1 | The address range that Address Range Comparator <m> defines, is selected. |

This bit is RES0 if $m \geq \text{TRCIDR4.NUMACPAIRS}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCBBCTLR

Must be programmed if `TRCCONFIGR.BB == 1`.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCBBCTLR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1111 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elseif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCBBCTLR;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elseif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCBBCTLR;
elseif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCBBCTLR;

```

MSR TRCBBCTLR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1111 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCBBCTLR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCBBCTLR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCBBCTLR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCCCTLR, Cycle Count Control Register

The TRCCCCTLR characteristics are:

Purpose

Set the threshold value for cycle counting.

Configuration

AArch64 System register TRCCCCTLR bits [31:0] are architecturally mapped to External register [TRCCCCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR0.TRCCCI == 1. Otherwise, direct accesses to TRCCCCTLR are UNDEFINED.

Attributes

TRCCCCTLR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | THRESHOLD | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:12]

Reserved, RES0.

THRESHOLD, bits [11:0]

Sets the threshold value for instruction trace cycle counting.

The minimum threshold value that can be programmed into THRESHOLD is given in [TRCIDR3.CCITMIN](#). If the THRESHOLD value is smaller than the value in [TRCIDR3.CCITMIN](#) then the behavior is CONSTRAINED UNPREDICTABLE. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

Writing a value of zero when [TRCCONFIGR.CCI](#) enables instruction trace cycle counting results in CONSTRAINED UNPREDICTABLE behavior. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCCCTLR

Must be programmed if [TRCCONFIGR.CCI](#) == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCCCCTLR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCCCTLR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCCCCTLR;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCCCTLR;

```

MSR TRCCCCTLR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCCCTLR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCCCTLR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCCCTLR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

Purpose

Configuration

Attributes

Field descriptions

Bits [63:32]

COMP3[<m>], bit [m+24], for m = 7 to 0
When TRCIDR4.NUMCIDC > 3:

This bit is RES0 if $m \geq \text{TRCIDR2.CIDSIZE}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP2[<m>], bit [m+16], for m = 7 to 0**When TRCIDR4.NUMCIDC > 2:**

TRCCIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR2. Each bit in this field corresponds to a byte in TRCCIDCVR2.

| COMP2[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR2[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR2[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP1[<m>], bit [m+8], for m = 7 to 0**When TRCIDR4.NUMCIDC > 1:**

TRCCIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR1. Each bit in this field corresponds to a byte in TRCCIDCVR1.

| COMP1[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR1[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR1[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP0[<m>], bit [m], for m = 7 to 0**When TRCIDR4.NUMCIDC > 0:**

TRCCIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR0. Each bit in this field corresponds to a byte in TRCCIDCVR0.

| COMP0[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR0[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR0[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCCIDCCTLR0

If software uses the [TRCCIDCVR<n>](#) registers, for n = 0 to 3, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in [TRCCIDCVR<n>](#) to 0x00.

If any bit is 1 and the relevant byte in [TRCCIDCVR<n>](#) is not 0x00, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCCIDCCTLR0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0011 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCIDCCTLR0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCCIDCCTLR0;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCIDCCTLR0;

```

MSR TRCCIDCCTLR0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0011 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCIDCCTLR0 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCIDCCTLR0 = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCIDCCTLR0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

Purpose

Configuration

Attributes

Field descriptions

Page 2216

COMP6[<m>], bit [m+16], for m = 7 to 0**When TRCIDR4.NUMCIDC > 6:**

TRCCIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR6. Each bit in this field corresponds to a byte in TRCCIDCVR6.

| COMP6[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP5[<m>], bit [m+8], for m = 7 to 0**When TRCIDR4.NUMCIDC > 5:**

TRCCIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR5. Each bit in this field corresponds to a byte in TRCCIDCVR5.

| COMP5[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP4[<m>], bit [m], for m = 7 to 0**When TRCIDR4.NUMCIDC > 4:**

TRCCIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR4. Each bit in this field corresponds to a byte in TRCCIDCVR4.

| COMP4[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCCIDCCTLR1

If software uses the [TRCCIDCVR<n>](#) registers, for n = 4 to 7, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in [TRCCIDCVR<n>](#) to 0x00.

If any bit is 1 and the relevant byte in [TRCCIDCVR<n>](#) is not 0x00, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCCIDCCTLR1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0011 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCIDCCTLR1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCCIDCCTLR1;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCIDCCTLR1;

```

MSR TRCCIDCCTLR1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0011 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCIDCCTLR1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCIDCCTLR1 = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCIDCCTLR1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDCVR<n>, Context Identifier Comparator Value Registers <n>, n = 0 - 7

The TRCCIDCVR<n> characteristics are:

Purpose

Contains a Context identifier value.

Configuration

AArch64 System register TRCCIDCVR<n> bits [63:0] are architecturally mapped to External register [TRCCIDCVR<n>\[63:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMCIDC > n. Otherwise, direct accesses to TRCCIDCVR<n> are UNDEFINED.

Attributes

TRCCIDCVR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

VALUE, bits [63:0]

Context identifier value. The width of this field is indicated by [TRCIDR2.CIDSIZE](#). Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Context identifier is zero until the PE updates the Context identifier.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCIDCVR<n>

Must be programmed if any of the following are true:

- [TRCRSCTLR<a>.GROUP](#) == 0b0110 and [TRCRSCTLR<a>.CID\[n\]](#) == 1.
- [TRCACATR<a>.CONTEXTTYPE](#) == 0b01 or 0b11 and [TRCACATR<a>.CONTEXT](#) == n.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCCIDCVR<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-------|
| 0b10 | 0b001 | 0b0011 | n[2:0]:0b0 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCIDCVR[UInt(CRm<3:1>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCIDCVR[UInt(CRm<3:1>)];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCCIDCVR[UInt(CRm<3:1>)];

```

MSR TRCCIDCVR<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-------|
| 0b10 | 0b001 | 0b0011 | n[2:0]:0b0 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCIDCVR[UInt(CRm<3:1>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCIDCVR[UInt(CRm<3:1>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCIDCVR[UInt(CRm<3:1>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCLAIMCLR, Claim Tag Clear Register

The TRCCLAIMCLR characteristics are:

Purpose

In conjunction with [TRCCLAIMSET](#), provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information, see the CoreSight Architecture Specification.

Configuration

AArch64 System register TRCCLAIMCLR bits [31:0] are architecturally mapped to External register [TRCCLAIMCLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCCLAIMCLR are UNDEFINED.

Attributes

TRCCLAIMCLR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | |
|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 |
| CLR[31] | CLR[30] | CLR[29] | CLR[28] | CLR[27] | CLR[26] | CLR[25] | CLR[24] | CLR[23] | CLR[22] | CLR[21] | CLR[20] | CLR[19] | CLR[18] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 |

Bits [63:32]

Reserved, RES0.

CLR[<m>], bit [m], for m = 31 to 0

Claim Tag Clear. Indicates the current status of Claim Tag bit <m>, and is used to clear Claim Tag bit <m> to 0.

| CLR[<m>] | Meaning |
|----------|---|
| 0b0 | On a read: Claim Tag bit <m> is not set. On a write: Ignored. |
| 0b1 | On a read: Claim Tag bit <m> is set. On a write: Clear Claim tag bit <m> to 0. |

The number of Claim Tag bits implemented is indicated in [TRCCLAIMSET](#).

This bit reads-as-zero and ignores writes if m > the number of Claim Tag bits.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to 0.

Access to this field is **W1C**.

Accessing TRCCLAIMCLR

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCCLAIMCLR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0111 | 0b1001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCLAIM == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCLAIMCLR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCCLAIMCLR;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCLAIMCLR;

```

MSR TRCCLAIMCLR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0111 | 0b1001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRCLAIM == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCLAIMCLR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCLAIMCLR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCLAIMCLR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCLAIMSET, Claim Tag Set Register

The TRCLAIMSET characteristics are:

Purpose

In conjunction with [TRCCLAIMCLR](#), provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information, see the CoreSight Architecture Specification.

Configuration

AArch64 System register TRCCLAIMSET bits [31:0] are architecturally mapped to External register [TRCCLAIMSET\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCCLAIMSET are UNDEFINED.

The number of claim tag bits implemented is IMPLEMENTATION DEFINED. Arm recommends that implementations support a minimum of four claim tag bits, that is, SET[3:0] reads as 0b1111.

Attributes

TRCLAIMSET is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 |
| | | | | | | | | | | | | | |
| SET[31] | SET[30] | SET[29] | SET[28] | SET[27] | SET[26] | SET[25] | SET[24] | SET[23] | SET[22] | SET[21] | SET[20] | SET[19] | SET[18] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 |

Bits [63:32]

Reserved, RES0.

SET[<m>], bit [m], for m = 31 to 0

Claim Tag Set. Indicates whether Claim Tag bit <m> is implemented, and is used to set Claim Tag bit <m> to 1.

| SET[<m>] | Meaning |
|----------|---|
| 0b0 | On a read: Claim Tag bit <m> is not implemented. On a write: Ignored. |
| 0b1 | On a read: Claim Tag bit <m> is implemented. On a write: Set Claim Tag bit <m> to 1. |

This bit reads-as-zero and ignores writes if $m >$ the number of Claim Tag bits.

Access to this field is **RAO/W1S**.

Accessing TRCCLAIMSET

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCCLAIMSET

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0111 | 0b1000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCLAIM == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCLAIMSET;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCLAIMSET;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCCLAIMSET;

```

MSR TRCCLAIMSET, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0111 | 0b1000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRCLAIM == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCLAIMSET = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCLAIMSET = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCLAIMSET = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCNTCTLR<n>, Counter Control Register <n>, n = 0 - 3

The TRCCNTCTLR<n> characteristics are:

Purpose

Controls the operation of Counter <n>.

Configuration

AArch64 System register TRCCNTCTLR<n> bits [31:0] are architecturally mapped to External register [TRCCNTCTLR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMCNTR > n. Otherwise, direct accesses to TRCCNTCTLR<n> are UNDEFINED.

Attributes

TRCCNTCTLR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|----------|---------|---------------|------|--------------|---------------|------|--------------|----|----|----|----|----|----|----|--|
| 6362616059585756555453525150 | | | | | | | | | | | | | | | | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | CNTCHAIN | RLDSELF | RLDEVENT_TYPE | RES0 | RLDEVENT_SEL | CNTEVENT_TYPE | RES0 | CNTEVENT_SEL | | | | | | | | |
| 3130292827262524232221201918 | | | | | | | | | | | | | | | | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | |

Bits [63:18]

Reserved, RES0.

CNTCHAIN, bit [17]

For TRCCNTCTLR3 and TRCCNTCTLR1, this field controls whether the Counter decrements when a reload event occurs for Counter <n-1>.

| CNTCHAIN | Meaning |
|----------|--|
| 0b0 | The Counter does not decrement when a reload event for Counter <n-1> occurs. |
| 0b1 | Counter <n> decrements when a reload event for Counter <n-1> occurs. This concatenates Counter <n> and Counter <n-1>, to provide a larger count value. |

CNTCHAIN is not implemented for TRCCNTCTLR0 and TRCCNTCTLR2.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RLDSELF, bit [16]

Controls whether a reload event occurs for the Counter, when the Counter reaches zero.

| RLDSELF | Meaning |
|----------------|--|
| 0b0 | Normal mode. The Counter is in Normal mode. |
| 0b1 | Self-reload mode. The Counter is in Self-reload mode. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RLDEVENT_TYPE, bit [15]

Chooses the type of Resource Selector.

Selects an event, that when it occurs causes a reload event for Counter <n>.

| RLDEVENT_TYPE | Meaning |
|----------------------|---|
| 0b0 | A single Resource Selector. TRCCNTCTLR<n>.RLDEVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCCNTCTLR<n>.RLDEVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCCNTCTLR<n>.RLDEVENT.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [14:13]

Reserved, RES0.

RLDEVENT_SEL, bits [12:8]

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTLR<n>.RLDEVENT.TYPE controls whether TRCCNTCTLR<n>.RLDEVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Selects an event, that when it occurs causes a reload event for Counter <n>.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

CNTEVENT_TYPE, bit [7]

Chooses the type of Resource Selector.

Selects an event, that when it occurs causes Counter <n> to decrement.

| CNTEVENT_TYPE | Meaning |
|---------------|---|
| 0b0 | A single Resource Selector. TRCCNTCTLR<n>.CNTEVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCCNTCTLR<n>.CNTEVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCCNTCTLR<n>.CNTEVENT.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

CNTEVENT_SEL, bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTLR<n>.CNTEVENT.TYPE controls whether TRCCNTCTLR<n>.CNTEVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Selects an event, that when it occurs causes Counter <n> to decrement.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTCTLR<n>

Must be programmed if [TRCRSCTLR<a>.GROUP == 0b0010](#) and [TRCRSCTLR<a>.COUNTERS\[n\] == 1](#).

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCCNTCTLR<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|-------|
| 0b10 | 0b001 | 0b0000 | 0b01:n[1:0] | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCNTCTLR[UInt(CRm<1:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCNTCTLR[UInt(CRm<1:0>)];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCCNTCTLR[UInt(CRm<1:0>)];

```

MSR TRCCNTCTLR<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|-------|
| 0b10 | 0b001 | 0b0000 | 0b01:n[1:0] | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCNTCTLR[UInt(CRm<1:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCNTCTLR[UInt(CRm<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCNTCTLR[UInt(CRm<1:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCNTRLDVR<n>, Counter Reload Value Register <n>, n = 0 - 3

The TRCCNTRLDVR<n> characteristics are:

Purpose

This sets or returns the reload count value for Counter <n>.

Configuration

AArch64 System register TRCCNTRLDVR<n> bits [31:0] are architecturally mapped to External register [TRCCNTRLDVR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMCNTR > n. Otherwise, direct accesses to TRCCNTRLDVR<n> are UNDEFINED.

Attributes

TRCCNTRLDVR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

VALUE, bits [15:0]

Contains the reload value for Counter <n>. When a reload event occurs for Counter <n> then the trace unit copies the VALUE<n> field into Counter <n>.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTRLDVR<n>

Must be programmed if [TRCRSCTLR<a>](#).GROUP == 0b0010 and [TRCRSCTLR<a>](#).COUNTERS[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCCNTRLDVR<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|-------|
| 0b10 | 0b001 | 0b0000 | 0b00:n[1:0] | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCNTRLDVR[UInt(CRm<1:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCNTRLDVR[UInt(CRm<1:0>)];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCCNTRLDVR[UInt(CRm<1:0>)];

```

MSR TRCCNTRLDVR<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|-------|
| 0b10 | 0b001 | 0b0000 | 0b00:n[1:0] | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCNTRLDVR[UInt(CRm<1:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCNTRLDVR[UInt(CRm<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCNTRLDVR[UInt(CRm<1:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCNTVR<n>, Counter Value Register <n>, n = 0 - 3

The TRCCNTVR<n> characteristics are:

Purpose

This sets or returns the value of Counter <n>.

Configuration

AArch64 System register TRCCNTVR<n> bits [31:0] are architecturally mapped to External register [TRCCNTVR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMCNTR > n. Otherwise, direct accesses to TRCCNTVR<n> are UNDEFINED.

Attributes

TRCCNTVR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

VALUE, bits [15:0]

Contains the count value of Counter.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTVR<n>

Must be programmed if [TRCRSCTLR<a>](#).GROUP == 0b0010 and [TRCRSCTLR<a>](#).COUNTERS[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCCNTVR<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|-------|
| 0b10 | 0b001 | 0b0000 | 0b10:n[1:0] | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCCNTVRn == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCNTVR[UInt(CRm<1:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCNTVR[UInt(CRm<1:0>)];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCCNTVR[UInt(CRm<1:0>)];

```

MSR TRCCNTVR<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|-------|
| 0b10 | 0b001 | 0b0000 | 0b10:n[1:0] | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRCCNTVRn == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCNTVR[UInt(CRm<1:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCNTVR[UInt(CRm<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCNTVR[UInt(CRm<1:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCONFIGR, Trace Configuration Register

The TRCCONFIGR characteristics are:

Purpose

Controls the tracing options.

Configuration

AArch64 System register TRCCONFIGR bits [31:0] are architecturally mapped to External register [TRCCONFIGR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCCONFIGR are UNDEFINED.

Attributes

TRCCONFIGR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|------|------|----|----|------|-----|------|-----|----|------|------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | VMIDOPT | QE | RSTS | RES0 | | | VMID | CID | RES0 | CCI | BB | RES0 | RES1 | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

VMIDOPT, bit [15]

When TRCIDR2.VMIDOPT == 0b01:

Virtual context identifier selection control.

| VMIDOPT | Meaning |
|---------|---|
| 0b0 | VTTBR_EL2 .VMID is used as the Virtual context identifier. |
| 0b1 | CONTEXTIDR_EL2 .PROCID is used as the Virtual context identifier. |

When TRCIDR2.VMIDOPT == 0b00:

Reserved, RES0.

Virtual context identifier selection control.

[VTTBR_EL2](#).VMID is used as the Virtual context identifier.

When TRCIDR2.VMIDOPT == 0b10:

Reserved, RES1.

Virtual context identifier selection control.

[CONTEXTIDR_EL2](#).PROCID is used as the Virtual context identifier.

Otherwise:

Reserved, RES0.

QE, bits [14:13]

When TRCIDR0.QSUPP == 0b01:

Q element generation control.

| QE | Meaning |
|------|---|
| 0b00 | Q elements are disabled. |
| 0b01 | Q elements with instruction counts are enabled. |
| | Q elements without instruction counts are disabled. |

All other values are reserved.

When TRCIDR0.QSUPP == 0b10:

Q element generation control.

| QE | Meaning |
|------|--|
| 0b00 | Q elements are disabled. |
| 0b11 | Q elements with instruction counts are enabled. |
| | Q elements without instruction counts are enabled. |

All other values are reserved.

When TRCIDR0.QSUPP == 0b11:

Q element generation control.

| QE | Meaning |
|------|---|
| 0b00 | Q elements are disabled. |
| 0b01 | Q elements with instruction counts are enabled. |
| | Q elements without instruction counts are disabled. |
| 0b11 | Q elements with instruction counts are enabled. |
| | Q elements without instruction counts are enabled. |

All other values are reserved.

Otherwise:

Reserved, RES0.

RS, bit [12]

When TRCIDR0.RETSTACK == 1:

Return stack control.

| RS | Meaning |
|-----|---------------------------|
| 0b0 | Return stack is disabled. |
| 0b1 | Return stack is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TS, bit [11]

When TRCIDR0.TSSIZE != 0b000000:

Global timestamp tracing control.

| TS | Meaning |
|-----|---------------------------------------|
| 0b0 | Global timestamp tracing is disabled. |
| 0b1 | Global timestamp tracing is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [10:8]

Reserved, RES0.

VMID, bit [7]

When TRCIDR2.VMIDSIZE != 0b000000:

Virtual context identifier tracing control.

| VMID | Meaning |
|------|---|
| 0b0 | Virtual context identifier tracing is disabled. |
| 0b1 | Virtual context identifier tracing is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CID, bit [6]

When TRCIDR2.CIDSIZE != 0b000000:

Context identifier tracing control.

| CID | Meaning |
|-----|---|
| 0b0 | Context identifier tracing is disabled. |
| 0b1 | Context identifier tracing is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [5]

Reserved, RES0.

CCI, bit [4]

When **TRCIDR0.TRCCCI == 1**:

Cycle counting instruction tracing control.

| CCI | Meaning |
|-----|---|
| 0b0 | Cycle counting instruction tracing is disabled. |
| 0b1 | Cycle counting instruction tracing is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BB, bit [3]

When **TRCIDR0.TRCBB == 1**:

Branch broadcasting control.

| BB | Meaning |
|-----|----------------------------------|
| 0b0 | Branch broadcasting is disabled. |
| 0b1 | Branch broadcasting is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [2:1]

Reserved, RES0.

Bit [0]

Reserved, RES1.

Accessing TRCCONFIGR

Must always be programmed.

TRCCONFIGR.QE must be set to 0b00 if TRCCONFIGR.BB is not 0.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCCONFIGR

| | | | | |
|-----|-----|-----|-----|-----|
| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0100 | 0b000 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCONFIGR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCCONFIGR;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCCONFIGR;

```

MSR TRCCONFIGR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCONFIGR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCCONFIGR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCCONFIGR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVARCH, Device Architecture Register

The TRCDEVARCH characteristics are:

Purpose

Provides discovery information for the component.
For additional information, see the CoreSight Architecture Specification.

Configuration

AArch64 System register TRCDEVARCH bits [31:0] are architecturally mapped to External register [TRCDEVARCH\[31:0\]](#).
This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCDEVARCH are UNDEFINED.

Attributes

TRCDEVARCH is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|---------|----------|----|----|----|---------|----|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ARCHITECT | | | | | | | | | | | PRESENT | REVISION | | | | ARCHVER | | | | ARCHPART | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

ARCHITECT, bits [31:21]

Architect. Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code.

| ARCHITECT | Meaning |
|---------------|--|
| 0b01000111011 | JEP106 continuation code 0x4, ID code 0x3B. Arm Limited. |

Other values are defined by the JEDEC JEP106 standard.

This field reads as 0x23B.

PRESENT, bit [20]

DEVARCH Present. Defines that the DEVARCH register is present.

| PRESENT | Meaning |
|---------|--|
| 0b0 | Device Architecture information not present. |
| 0b1 | Device Architecture information present. |

This field reads as 1.

REVISION, bits [19:16]

Revision. Defines the architecture revision of the component.

| REVISION | Meaning |
|----------|------------------------|
| 0b0000 | ETEv1.0, FEAT_ETE. |
| 0b0001 | ETEv1.1, FEAT_ETEv1p1. |
| 0b0010 | ETEv1.2, FEAT_ETEv1p2. |

All other values are reserved.

ARCHVER, bits [15:12]

Architecture Version. Defines the architecture version of the component.

| ARCHVER | Meaning |
|---------|---------|
| 0b0101 | ETEv1. |

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHVER is ARCHID[15:12].

This field reads as 0x5.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

| ARCHPART | Meaning |
|----------|----------------------------|
| 0xA13 | Arm PE trace architecture. |

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHPART is ARCHID[11:0].

This field reads as 0xA13.

Accessing TRCDEVARCH

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCDEVARCH

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0111 | 0b1111 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCDEVARCH;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCDEVARCH;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCDEVARCH;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVID, Device Configuration Register

The TRCDEVID characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

AArch64 System register TRCDEVID bits [31:0] are architecturally mapped to External register [TRCDEVID\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCDEVID are UNDEFINED.

Attributes

TRCDEVID is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, RES0.

Accessing TRCDEVID

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCDEVID

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0111 | 0b0010 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCDEVID;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCDEVID;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCDEVID;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCEVENTCTL0R, Event Control 0 Register

The TRCEVENTCTL0R characteristics are:

Purpose

Controls the generation of ETEEvents.

Configuration

AArch64 System register TRCEVENTCTL0R bits [31:0] are architecturally mapped to External register [TRCEVENTCTL0R\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMRSPAIR != 0b0000. Otherwise, direct accesses to TRCEVENTCTL0R are UNDEFINED.

Attributes

TRCEVENTCTL0R is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|------|------------|-------------|------|------------|-------------|------|------------|-------------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| EVENT3_TYPE | RES0 | EVENT3_SEL | EVENT2_TYPE | RES0 | EVENT2_SEL | EVENT1_TYPE | RES0 | EVENT1_SEL | EVENT0_TYPE | RES0 | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | |

Bits [63:32]

Reserved, RES0.

EVENT3_TYPE, bit [31]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b11:

Chooses the type of Resource Selector.

| EVENT3_TYPE | Meaning |
|-------------|---|
| 0b0 | A single Resource Selector. TRCEVENTCTL0R.EVENT3.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT3.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT3.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [30:29]

Reserved, RES0.

EVENT3_SEL, bits [28:24]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b11:

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT3.TYPE controls whether TRCEVENTCTL0R.EVENT3.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

When any of the selected resource events occurs and [TRCEVENTCTL1R](#).INSTEN[3] == 1, then Event element 3 is generated in the instruction trace element stream.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT2_TYPE, bit [23]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b10:

Chooses the type of Resource Selector.

| EVENT2_TYPE | Meaning |
|-------------|---|
| 0b0 | A single Resource Selector. TRCEVENTCTL0R.EVENT2.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT2.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [22:21]

Reserved, RES0.

EVENT2_SEL, bits [20:16]**When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b10:**

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.TYPE controls whether TRCEVENTCTL0R.EVENT2.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

When any of the selected resource events occurs and [TRCEVENTCTL1R](#).INSTEN[2] == 1, then Event element 2 is generated in the instruction trace element stream.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT1_TYPE, bit [15]**When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b01:**

Chooses the type of Resource Selector.

| EVENT1_TYPE | Meaning |
|-------------|---|
| 0b0 | A single Resource Selector. TRCEVENTCTL0R.EVENT1.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT1.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [14:13]

Reserved, RES0.

EVENT1_SEL, bits [12:8]**When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b01:**

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.TYPE controls whether TRCEVENTCTL0R.EVENT1.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

When any of the selected resource events occurs and [TRCEVENTCTL1R](#).INSTEN[1] == 1, then Event element 1 is generated in the instruction trace element stream.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT0_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b00:

Chooses the type of Resource Selector.

| EVENT0_TYPE | Meaning |
|-------------|---|
| 0b0 | A single Resource Selector. TRCEVENTCTL0R.EVENT0.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT0.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT0.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EVENT0_SEL, bits [4:0]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b00:

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT0.TYPE controls whether TRCEVENTCTL0R.EVENT0.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

When any of the selected resource events occurs and [TRCEVENTCTL1R](#).INSTEN[0] == 1, then Event element 0 is generated in the instruction trace element stream.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCEVENTCTL0R

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCEVENTCTL0R

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCEVENTCTL0R;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCEVENTCTL0R;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCEVENTCTL0R;

```

MSR TRCEVENTCTL0R, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCEVENTCTL0R = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCEVENTCTL0R = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCEVENTCTL0R = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCEVENTCTL1R, Event Control 1 Register

The TRCEVENTCTL1R characteristics are:

Purpose

Controls the behavior of the ETEEvents that [TRCEVENTCTL0R](#) selects.

Configuration

AArch64 System register TRCEVENTCTL1R bits [31:0] are architecturally mapped to External register [TRCEVENTCTL1R\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCEVENTCTL1R are UNDEFINED.

Attributes

TRCEVENTCTL1R is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|-----|----|------|----|----|----|----|----|-----------|----|-----------|----|-----------|----|-----------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | LPOVERRIDE | | ATB | | RES0 | | | | | | INSTEN[3] | | INSTEN[2] | | INSTEN[1] | | INSTEN[0] | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:13]

Reserved, RES0.

LPOVERRIDE, bit [12]

When TRCIDR5.LPOVERRIDE == 1:

Low-power Override Mode select.

| LPOVERRIDE | Meaning |
|------------|--|
| 0b0 | Trace unit Low-power Override Mode is not enabled. That is, the trace unit is permitted to enter low-power state. |
| 0b1 | Trace unit Low-power Override Mode is enabled. That is, entry to a low-power state does not affect the trace unit resources or trace generation. |

Otherwise:

Reserved, RES0.

ATB, bit [11]

When TRCIDR5.ATBTRIG == 1:

AMBA Trace Bus (ATB) trigger enable.

If a CoreSight ATB interface is implemented then when ETEEvent 0 occurs the trace unit sets:

- ATID == 0x7D.

- ATDATA to the value of [TRCTRACEIDR](#).

If the width of ATDATA is greater than the width of [TRCTRACEIDR](#).TRACEID then the trace unit zeros the upper ATDATA bits.

If ETEEvent 0 is programmed to occur based on program execution, such as an Address Comparator, the ATB trigger might not be inserted into the ATB stream at the same time as any trace generated by that program execution is output by the trace unit. Typically, the generated trace might be buffered in a trace unit which means that the ATB trigger would be output before the associated trace is output.

If ETEEvent 0 is asserted multiple times in close succession, the trace unit is required to generate an ATB trigger for the first assertion, but might ignore one or more of the subsequent assertions. Arm recommends that the window in which ETEEvent 0 is ignored is limited only by the time taken to output an ATB trigger.

| ATB | Meaning |
|-----|--------------------------|
| 0b0 | ATB trigger is disabled. |
| 0b1 | ATB trigger is enabled. |

Otherwise:

Reserved, RES0.

Bits [10:4]

Reserved, RES0.

INSTEN[<m>], bit [m], for m = 3 to 0

Event element control.

| INSTEN[<m>] | Meaning |
|-------------|--|
| 0b0 | The trace unit does not generate an Event element <m>. |
| 0b1 | The trace unit generates an Event element <m>. |

This bit is RES0 if m >= the number indicated by [TRCIDR0](#).NUMEVENT.

Accessing TRCEVENTCTL1R

Must be programmed.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCEVENTCTL1R

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCEVENTCTL1R;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCEVENTCTL1R;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCEVENTCTL1R;

```

MSR TRCEVENTCTL1R, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCEVENTCTL1R = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCEVENTCTL1R = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCEVENTCTL1R = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCEXTINSELN<n>, External Input Select Register <n>, n = 0 - 3

The TRCEXTINSELN<n> characteristics are:

Purpose

Use this to set, or read, which External Inputs are resources to the trace unit.

The name TRCEXTINSELN is an alias of TRCEXTINSELN0.

Configuration

AArch64 System register TRCEXTINSELN<n> bits [31:0] are architecturally mapped to External register [TRCEXTINSELN<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMEXTINSEL > n. Otherwise, direct accesses to TRCEXTINSELN<n> are UNDEFINED.

Attributes

TRCEXTINSELN<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | evtCount | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

evtCount, bits [15:0]

PMU event to select.

The event number as defined by the Arm ARM.

Software must program this field with a PMU event that is supported by the PE being programmed.

There are three ranges of PMU event numbers:

- PMU event numbers in the range 0x0000 to 0x003F are common architectural and microarchitectural events.
- PMU event numbers in the range 0x0040 to 0x00BF are Arm recommended common architectural and microarchitectural PMU events.
- PMU event numbers in the range 0x00C0 to 0x03FF are IMPLEMENTATION DEFINED PMU events.

If evtCount is programmed to a PMU event that is reserved or not supported by the PE, the behavior depends on the PMU event type:

- For the range 0x0000 to 0x003F, then the PMU event is not active, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- For IMPLEMENTATION DEFINED PMU events, it is UNPREDICTABLE what PMU event, if any, is counted, and the value returned by a direct or external read of the evtCount field is UNKNOWN.

UNPREDICTABLE means the PMU event must not expose privileged information.

Arm recommends that the behavior across a family of implementations is defined such that if a given implementation does not include a PMU event from a set of common IMPLEMENTATION_DEFINED PMU events, then no PMU event is counted and the value read back on evtCount is the value written.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCEXTINSELN<n>

Must be programmed if any of the following is true: [TRCRSCTLR<a>](#).GROUP == 0b0000 and [TRCRSCTLR<a>](#).EXTIN[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCEXTINSELN<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|-------|
| 0b10 | 0b001 | 0b0000 | 0b10:n[1:0] | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elseif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCEXTINSELN[UInt(CRm<1:0>)];
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elseif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCEXTINSELN[UInt(CRm<1:0>)];
elseif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCEXTINSELN[UInt(CRm<1:0>)];

```

MSR TRCEXTINSELN<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|-------|
| 0b10 | 0b001 | 0b0000 | 0b10:n[1:0] | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCEXTINSELN[UInt(CRm<1:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCEXTINSELN[UInt(CRm<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCEXTINSELN[UInt(CRm<1:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR0, ID Register 0

The TRCIDR0 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR0 bits [31:0] are architecturally mapped to External register [TRCIDR0\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR0 are UNDEFINED.

Attributes

TRCIDR0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----------|---------|--------|--------|------|-----------|-------|-------|----------|----------|----------|----|----|----|----|----|----|------|----|----|----|----|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | | | | |
| | | | | | | | | | | | | | | | | | | RES0 | | | | | | | | |
| RES0 | COMMTRANS | COMMOPT | TSSIZE | TSMARK | RES0 | TRCEXDATA | QSUPP | QFILT | CONDTYPE | NUMEVENT | RETSTACK | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | | | | |

Bits [63:31]

Reserved, RES0.

COMMTRANS, bit [30]

Transaction Start element behavior.

| COMMTRANS | Meaning |
|-----------|---|
| 0b0 | Transaction Start elements are P0 elements. |
| 0b1 | Transaction Start elements are not P0 elements. |

COMMOPT, bit [29]

Indicates the contents and encodings of Cycle count packets.

| COMMOPT | Meaning |
|---------|----------------|
| 0b0 | Commit mode 0. |
| 0b1 | Commit mode 1. |

The Commit mode defines the contents and encodings of Cycle Count packets, in particular how Commit elements are indicated by these packets. See the descriptions of these packets for more details.

Accessing this field has the following behavior:

- **RAO/WI** if all of the following are true:
 - TRCIDR0.TRCCCI == 1
 - TRCIDR8.MAXSPEC == 0x0
- When TRCIDR0.TRCCCI == 0, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RO**.

TSSIZE, bits [28:24]

Indicates that the trace unit implements Global timestamping and the size of the timestamp value.

| TSSIZE | Meaning |
|---------------|--|
| 0b00000 | Global timestamping not implemented. |
| 0b01000 | Global timestamping implemented with a 64-bit timestamp value. |

All other values are reserved.

This field reads as 0b01000.

TSMARK, bit [23]

When FEAT_ETEv1p1 is implemented:

Indicates whether Timestamp Marker elements are generated.

| TSMARK | Meaning |
|---------------|--|
| 0b0 | Timestamp Marker elements are not generated. |
| 0b1 | Timestamp Marker elements are generated. |

Otherwise:

Reserved, RES0.

Bits [22:18]

Reserved, RES0.

TRCEXDATA, bit [17]

When TRCIDR0.TRCDATA != 0b00:

Indicates if the trace unit implements tracing of data transfers for exceptions and exception returns. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

| TRCEXDATA | Meaning |
|------------------|---|
| 0b0 | Tracing of data transfers for exceptions and exception returns not implemented. |
| 0b1 | Tracing of data transfers for exceptions and exception returns implemented. |

Otherwise:

Reserved, RES0.

QSUPP, bits [16:15]

Indicates that the trace unit implements Q element support.

| QSUPP | Meaning |
|--------------|---|
| 0b00 | Q element support is not implemented. |
| 0b01 | Q element support is implemented, and only supports Q elements with instruction counts. |
| 0b10 | Q element support is implemented, and only supports Q elements without instruction counts. |
| 0b11 | Q element support is implemented, and supports: <ul style="list-style-type: none"> Q elements with instruction counts. Q elements without instruction counts. |

QFILT, bit [14]

Indicates if the trace unit implements Q element filtering.

| QFILT | Meaning |
|-------|---|
| 0b0 | Q element filtering is not implemented. |
| 0b1 | Q element filtering is implemented. |

If TRCIDR0.QSUPP == 0b00 then this field is 0.

CONDTYPE, bits [13:12]

When TRCIDR0.TRCCOND == 1:

Indicates how conditional instructions are traced. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

| CONDTYPE | Meaning |
|----------|---|
| 0b00 | Conditional instructions are traced with an indication of whether they pass or fail their condition code check. |
| 0b01 | Conditional instructions are traced with an indication of the APSR condition flags. |

All other values are reserved.

Otherwise:

Reserved, RES0.

NUMEVENT, bits [11:10]

When TRCIDR4.NUMRSPAIR == 0b0000:

Indicates the number of ETEEvents implemented.

| NUMEVENT | Meaning |
|----------|--------------------------------------|
| 0b00 | The trace unit supports 0 ETEEvents. |

All other values are reserved.

When TRCIDR4.NUMRSPAIR != 0b0000:

Indicates the number of ETEEvents implemented.

| NUMEVENT | Meaning |
|----------|--------------------------------------|
| 0b00 | The trace unit supports 1 ETEEvent. |
| 0b01 | The trace unit supports 2 ETEEvents. |
| 0b10 | The trace unit supports 3 ETEEvents. |
| 0b11 | The trace unit supports 4 ETEEvents. |

Otherwise:

Reserved, RES0.

RETSTACK, bit [9]

Indicates if the trace unit supports the return stack.

| RETSTACK | Meaning |
|----------|-------------------------------|
| 0b0 | Return stack not implemented. |
| 0b1 | Return stack implemented. |

Bit [8]

Reserved, RES0.

TRCCCI, bit [7]

Indicates if the trace unit implements cycle counting.

| TRCCCI | Meaning |
|--------|---------------------------------|
| 0b0 | Cycle counting not implemented. |
| 0b1 | Cycle counting implemented. |

This field reads as 1.

TRCCOND, bit [6]

Indicates if the trace unit implements conditional instruction tracing. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures.

| TRCCOND | Meaning |
|---------|--|
| 0b0 | Conditional instruction tracing not implemented. |
| 0b1 | Conditional instruction tracing implemented. |

This field reads as 0.

TRCBB, bit [5]

Indicates if the trace unit implements branch broadcasting.

| TRCBB | Meaning |
|-------|--------------------------------------|
| 0b0 | Branch broadcasting not implemented. |
| 0b1 | Branch broadcasting implemented. |

This field reads as 1.

TRCDATA, bits [4:3]

Indicates if the trace unit implements data tracing. Data tracing is not implemented in ETE and this field is reserved for other trace architectures.

| TRCDATA | Meaning |
|---------|-------------------------------|
| 0b00 | Data tracing not implemented. |
| 0b11 | Data tracing implemented. |

All other values are reserved.

This field reads as 0b00.

INSTP0, bits [2:1]

Indicates if load and store instructions are P0 instructions. Load and store instructions as P0 instructions is not implemented in ETE and this field is reserved for other trace architectures.

| INSTP0 | Meaning |
|--------|--|
| 0b00 | Load and store instructions are not P0 instructions. |
| 0b11 | Load and store instructions are P0 instructions. |

All other values are reserved.

This field reads as 0b00.

Bit [0]

Reserved, RES1.

Accessing TRCIDR0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR0;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR0;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR1, ID Register 1

The TRCIDR1 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR1 bits [31:0] are architecturally mapped to External register [TRCIDR1\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR1 are UNDEFINED.

Attributes

TRCIDR1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|------------|----|----|----|------------|----|----|----|----------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| DESIGNER | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| DESIGNER | | | | | | | | RES0 | | | | | | | | RES1 | | | | TRCARCHMAJ | | | | TRCARCHMIN | | | | REVISION | | | |

Bits [63:32]

Reserved, RES0.

DESIGNER, bits [31:24]

Indicates which company designed the trace unit. The permitted values of this field are the same as [MIDR_EL1.Implementer](#).

Bits [23:16]

Reserved, RES0.

Bits [15:12]

Reserved, RES1.

TRCARCHMAJ, bits [11:8]

Major architecture version.

| TRCARCHMAJ | Meaning |
|------------|---|
| 0b1111 | If both TRCARCHMAJ and TRCARCHMIN == 0xF then refer to TRCDEVARCH . |

All other values are reserved.

This field reads as 0b1111.

TRCARCHMIN, bits [7:4]

Minor architecture version.

| TRCARCHMIN | Meaning |
|------------|---|
| 0b1111 | If both TRCARCHMAJ and TRCARCHMIN == 0xF then refer to TRCDEVARCH . |

All other values are reserved.

This field reads as 0b1111.

REVISION, bits [3:0]

Implementation revision.

Returns an IMPLEMENTATION DEFINED value that identifies the revision of the trace unit.

Arm deprecates any use of this field and recommends that implementations set this field to zero.

Accessing TRCIDR1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1001 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCIDR1;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR10, ID Register 10

The TRCIDR10 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

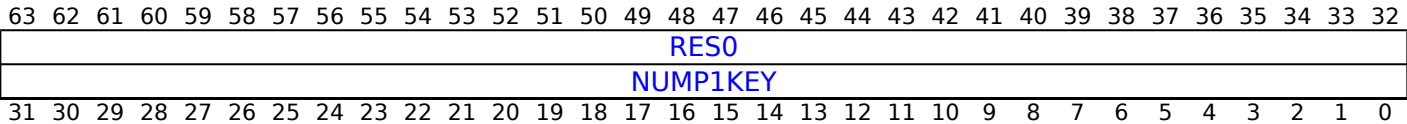
AArch64 System register TRCIDR10 bits [31:0] are architecturally mapped to External register [TRCIDR10\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR10 are UNDEFINED.

Attributes

TRCIDR10 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

NUMP1KEY, bits [31:0] When TRCIDR0.TRCDATA != 0b00:

Indicates the number of P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

Accessing TRCIDR10

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR10

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR10;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCIDR10;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR10;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR11, ID Register 11

The TRCIDR11 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR11 bits [31:0] are architecturally mapped to External register [TRCIDR11\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR11 are UNDEFINED.

Attributes

TRCIDR11 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | NUMP1SPC | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

NUMP1SPC, bits [31:0] When TRCIDR0.TRCDATA != 0b00:

Indicates the number of special P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

Accessing TRCIDR11

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR11

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0011 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR11;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCIDR11;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR11;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR12, ID Register 12

The TRCIDR12 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

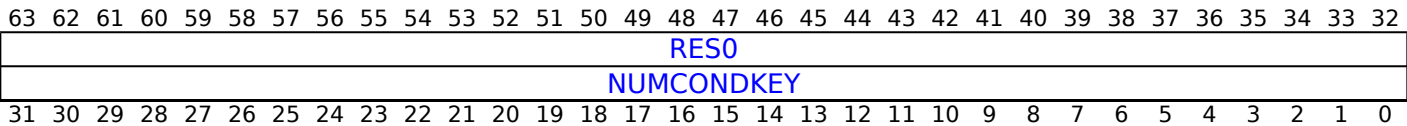
AArch64 System register TRCIDR12 bits [31:0] are architecturally mapped to External register [TRCIDR12\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR12 are UNDEFINED.

Attributes

TRCIDR12 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

NUMCONDKEY, bits [31:0] When TRCIDR0.TRCCOND == 1:

Indicates the number of conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

Accessing TRCIDR12

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0100 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR12;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCIDR12;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR12;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR13, ID Register 13

The TRCIDR13 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

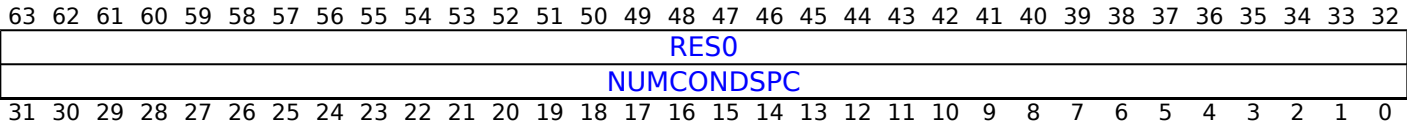
AArch64 System register TRCIDR13 bits [31:0] are architecturally mapped to External register [TRCIDR13\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR13 are UNDEFINED.

Attributes

TRCIDR13 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

NUMCONDSPC, bits [31:0] When TRCIDR0.TRCCOND == 1:

Indicates the number of special conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

Accessing TRCIDR13

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR13

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0101 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR13;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCIDR13;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR13;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR2, ID Register 2

The TRCIDR2 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR2 bits [31:0] are architecturally mapped to External register [TRCIDR2\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR2 are UNDEFINED.

Attributes

TRCIDR2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|---------|--------|----|----|----|--------|----|----|----|--------|----|----|----|----------|----|----|----|---------|----|----|----|--------|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| WFXMODE | VMIDOPT | CCSIZE | | | | DVSIZE | | | | DASIZE | | | | VMIDSIZE | | | | CIDSIZE | | | | IASIZE | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

WFXMODE, bit [31]

Indicates whether WFI and WFE instructions are classified as P0 instructions:

| WFXMODE | Meaning |
|---------|---|
| 0b0 | WFI and WFE instructions are not classified as P0 instructions. |
| 0b1 | WFI and WFE instructions are classified as P0 instructions. |

VMIDOPT, bits [30:29]

Indicates the options for Virtual context identifier selection.

| VMIDOPT | Meaning |
|---------|---|
| 0b00 | Virtual context identifier selection not supported. |
| 0b01 | TRCCONFIGR .VMIDOPT is RES0. |
| 0b10 | Virtual context identifier selection supported. |
| 0b11 | TRCCONFIGR .VMIDOPT is implemented. |
| 0b10 | Virtual context identifier selection not supported. |
| 0b11 | TRCCONFIGR .VMIDOPT is RES1. |

All other values are reserved.

If TRCIDR2.VMIDSIZE == 0b000000 then this field is 0b00.

If TRCIDR2.VMIDSIZE != 0b000000 then this field is 0b10.

CCSIZE, bits [28:25]**When TRCIDR0.TRCCCI == 1:**

Indicates the size of the cycle counter.

| CCSIZE | Meaning |
|---------------|---|
| 0b0000 | The cycle counter is 12 bits in length. |
| 0b0001 | The cycle counter is 13 bits in length. |
| 0b0010 | The cycle counter is 14 bits in length. |
| 0b0011 | The cycle counter is 15 bits in length. |
| 0b0100 | The cycle counter is 16 bits in length. |
| 0b0101 | The cycle counter is 17 bits in length. |
| 0b0110 | The cycle counter is 18 bits in length. |
| 0b0111 | The cycle counter is 19 bits in length. |
| 0b1000 | The cycle counter is 20 bits in length. |

All other values are reserved.

Otherwise:

Reserved, RES0.

DVSIZE, bits [24:20]**When TRCIDR0.TRCDATA != 0b00:**

Indicates the data value size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

| DVSIZE | Meaning |
|---------------|---|
| 0b00000 | Data value tracing not implemented. |
| 0b00100 | Data value tracing has a maximum of 32-bit data values. |
| 0b01000 | Data value tracing has a maximum of 64-bit data values. |

All other values are reserved.

Otherwise:

Reserved, RES0.

DASIZE, bits [19:15]**When TRCIDR0.TRCDATA != 0b00:**

Indicates the data address size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

| DASIZE | Meaning |
|---------------|--|
| 0b00000 | Data address tracing not implemented. |
| 0b00100 | Data address tracing has a maximum of 32-bit data addresses. |
| 0b01000 | Data address tracing has a maximum of 64-bit data addresses. |

All other values are reserved.

Otherwise:

Reserved, RES0.

VMIDSIZE, bits [14:10]

Indicates the trace unit Virtual context identifier size.

| VMIDSIZE | Meaning |
|-----------------|--|
| 0b000000 | Virtual context identifier tracing is not supported. |
| 0b000001 | 8-bit Virtual context identifier size. |
| 0b000010 | 16-bit Virtual context identifier size. |
| 0b000100 | 32-bit Virtual context identifier size. |

All other values are reserved.

If the PE does not implement EL2 then this field is 0b000000.

If the PE implements EL2 then this field is 0b000100.

CIDSIZE, bits [9:5]

Indicates the Context identifier size.

| CIDSIZE | Meaning |
|----------------|--|
| 0b000000 | Context identifier tracing is not supported. |
| 0b000100 | 32-bit Context identifier size. |

All other values are reserved.

This field reads as 0b000100.

IASIZE, bits [4:0]

Virtual instruction address size.

| IASIZE | Meaning |
|---------------|---|
| 0b000100 | Maximum of 32-bit instruction address size. |
| 0b010000 | Maximum of 64-bit instruction address size. |

All other values are reserved.

This field reads as 0b010000.

Accessing TRCIDR2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR2

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b10 | 0b001 | 0b0000 | 0b1010 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR2;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCIDR2;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR2;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

| SYSSTALL | Meaning |
|-----------------|--------------------------------------|
| 0b0 | Stalling of the PE is not permitted. |
| 0b1 | Stalling of the PE is permitted. |

The value of this field might be dynamic and change based on system conditions.

If TRCIDR3.STALLCTL == 0 then this field is 0.

STALLCTL, bit [26]

Indicates if trace unit implements stalling of the PE.

| STALLCTL | Meaning |
|-----------------|--|
| 0b0 | Stalling of the PE is not implemented. |
| 0b1 | Stalling of the PE is implemented. |

SYNCPR, bit [25]

Indicates if an implementation has a fixed synchronization period.

| SYNCPR | Meaning |
|---------------|--|
| 0b0 | TRCSYNCPR is read-write so software can change the synchronization period. |
| 0b1 | TRCSYNCPR is read-only so the synchronization period is fixed. |

This field reads as 0.

TRCERR, bit [24]

Indicates forced tracing of System Error exceptions is implemented.

| TRCERR | Meaning |
|---------------|---|
| 0b0 | Forced tracing of System Error exceptions is not implemented. |
| 0b1 | Forced tracing of System Error exceptions is implemented. |

This field reads as 1.

Bit [23]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [22]

Indicates if Non-secure EL2 is implemented.

| EXLEVEL_NS_EL2 | Meaning |
|-----------------------|------------------------------------|
| 0b0 | Non-secure EL2 is not implemented. |
| 0b1 | Non-secure EL2 is implemented. |

EXLEVEL_NS_EL1, bit [21]

Indicates if Non-secure EL1 is implemented.

| EXLEVEL_NS_EL1 | Meaning |
|-----------------------|------------------------------------|
| 0b0 | Non-secure EL1 is not implemented. |
| 0b1 | Non-secure EL1 is implemented. |

EXLEVEL_NS_EL0, bit [20]

Indicates if Non-secure EL0 is implemented.

| EXLEVEL_NS_EL0 | Meaning |
|----------------|------------------------------------|
| 0b0 | Non-secure EL0 is not implemented. |
| 0b1 | Non-secure EL0 is implemented. |

EXLEVEL_S_EL3, bit [19]

Indicates if EL3 is implemented.

| EXLEVEL_S_EL3 | Meaning |
|---------------|-------------------------|
| 0b0 | EL3 is not implemented. |
| 0b1 | EL3 is implemented. |

EXLEVEL_S_EL2, bit [18]

Indicates if Secure EL2 is implemented.

| EXLEVEL_S_EL2 | Meaning |
|---------------|--------------------------------|
| 0b0 | Secure EL2 is not implemented. |
| 0b1 | Secure EL2 is implemented. |

EXLEVEL_S_EL1, bit [17]

Indicates if Secure EL1 is implemented.

| EXLEVEL_S_EL1 | Meaning |
|---------------|--------------------------------|
| 0b0 | Secure EL1 is not implemented. |
| 0b1 | Secure EL1 is implemented. |

EXLEVEL_S_EL0, bit [16]

Indicates if Secure EL0 is implemented.

| EXLEVEL_S_EL0 | Meaning |
|---------------|--------------------------------|
| 0b0 | Secure EL0 is not implemented. |
| 0b1 | Secure EL0 is implemented. |

Bits [15:14]

Reserved, RES0.

CCITMIN, bits [11:0]

Indicates the minimum value that can be programmed in [TRCCCCTLR](#).THRESHOLD.

If [TRCIDR0](#).TRCCCI == 1 then the minimum value of this field is 0x001.

If [TRCIDR0](#).TRCCCI == 0 then this field is zero.

Accessing TRCIDR3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1011 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR3;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCIDR3;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR3;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR4, ID Register 4

The TRCIDR4 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR4 bits [31:0] are architecturally mapped to External register [TRCIDR4\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR4 are UNDEFINED.

Attributes

TRCIDR4 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|---------|----|----|----|---------|----|----|----|-----------|----|----|----|-------|----|----|----|------|----|----|--|---------|--|----|----|--------|----|----|----|------------|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | | 40 | | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| NUMVMIDC | | | | NUMCIDC | | | | NUMSSCC | | | | NUMRSPAIR | | | | NUMPC | | | | RES0 | | | | SUPPDAC | | | | NUMDVC | | | | NUMACPAIRS | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | | 8 | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:32]

Reserved, RES0.

NUMVMIDC, bits [31:28]

Indicates the number of Virtual Context Identifier Comparators that are available for tracing.

| NUMVMIDC | Meaning |
|----------|--|
| 0b0000 | No Virtual Context Identifier Comparators are available. |
| 0b0001 | The implementation has one Virtual Context Identifier Comparator. |
| 0b0010 | The implementation has two Virtual Context Identifier Comparators. |
| 0b0011 | The implementation has three Virtual Context Identifier Comparators. |
| 0b0100 | The implementation has four Virtual Context Identifier Comparators. |
| 0b0101 | The implementation has five Virtual Context Identifier Comparators. |
| 0b0110 | The implementation has six Virtual Context Identifier Comparators. |
| 0b0111 | The implementation has seven Virtual Context Identifier Comparators. |
| 0b1000 | The implementation has eight Virtual Context Identifier Comparators. |

All other values are reserved.

NUMCIDC, bits [27:24]

Indicates the number of Context Identifier Comparators that are available for tracing.

| NUMC IDC | Meaning |
|----------|--|
| 0b0000 | No Context Identifier Comparators are available. |
| 0b0001 | The implementation has one Context Identifier Comparator. |
| 0b0010 | The implementation has two Context Identifier Comparators. |
| 0b0011 | The implementation has three Context Identifier Comparators. |
| 0b0100 | The implementation has four Context Identifier Comparators. |
| 0b0101 | The implementation has five Context Identifier Comparators. |
| 0b0110 | The implementation has six Context Identifier Comparators. |
| 0b0111 | The implementation has seven Context Identifier Comparators. |
| 0b1000 | The implementation has eight Context Identifier Comparators. |

All other values are reserved.

NUMSSCC, bits [23:20]

Indicates the number of Single-shot Comparator Controls that are available for tracing.

| NUMSSCC | Meaning |
|---------|---|
| 0b0000 | No Single-shot Comparator Controls are available. |
| 0b0001 | The implementation has one Single-shot Comparator Control. |
| 0b0010 | The implementation has two Single-shot Comparator Controls. |
| 0b0011 | The implementation has three Single-shot Comparator Controls. |
| 0b0100 | The implementation has four Single-shot Comparator Controls. |
| 0b0101 | The implementation has five Single-shot Comparator Controls. |
| 0b0110 | The implementation has six Single-shot Comparator Controls. |
| 0b0111 | The implementation has seven Single-shot Comparator Controls. |
| 0b1000 | The implementation has eight Single-shot Comparator Controls. |

All other values are reserved.

NUMRSPAIR, bits [19:16]

Indicates the number of resource selector pairs that are available for tracing.

| NUMRSPAIR | Meaning |
|-----------|--|
| 0b0000 | The implementation has zero resource selectors. |
| 0b0001 | The implementation has two resource selector pairs. |
| 0b0010 | The implementation has three resource selector pairs. |
| 0b0011 | The implementation has four resource selector pairs. |
| 0b0100 | The implementation has five resource selector pairs. |
| 0b0101 | The implementation has six resource selector pairs. |
| 0b0110 | The implementation has seven resource selector pairs. |
| 0b0111 | The implementation has eight resource selector pairs. |
| 0b1000 | The implementation has nine resource selector pairs. |
| 0b1001 | The implementation has ten resource selector pairs. |
| 0b1010 | The implementation has eleven resource selector pairs. |
| 0b1011 | The implementation has twelve resource selector pairs. |
| 0b1100 | The implementation has thirteen resource selector pairs. |
| 0b1101 | The implementation has fourteen resource selector pairs. |
| 0b1110 | The implementation has fifteen resource selector pairs. |
| 0b1111 | The implementation has sixteen resource selector pairs. |

All other values are reserved.

NUMPC, bits [15:12]

Indicates the number of PE Comparator Inputs that are available for tracing.

| NUMPC | Meaning |
|--------|--|
| 0b0000 | No PE Comparator Inputs are available. |
| 0b0001 | The implementation has one PE Comparator Input. |
| 0b0010 | The implementation has two PE Comparator Inputs. |
| 0b0011 | The implementation has three PE Comparator Inputs. |
| 0b0100 | The implementation has four PE Comparator Inputs. |
| 0b0101 | The implementation has five PE Comparator Inputs. |
| 0b0110 | The implementation has six PE Comparator Inputs. |
| 0b0111 | The implementation has seven PE Comparator Inputs. |
| 0b1000 | The implementation has eight PE Comparator Inputs. |

All other values are reserved.

Bits [11:9]

Reserved, RES0.

SUPPDAC, bit [8]

When TRCIDR4.NUMACPAIRS != 0b0000:

Indicates whether data address comparisons are implemented. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

| SUPPDAC | Meaning |
|---------|---|
| 0b0 | Data address comparisons not implemented. |
| 0b1 | Data address comparisons implemented. |

This field reads as 0.

Otherwise:

Reserved, RES0.

NUMDVC, bits [7:4]

Indicates the number of data value comparators. Data value comparators are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

| NUMDVC | Meaning |
|--------|---|
| 0b0000 | No data value comparators implemented. |
| 0b0001 | One data value comparator implemented. |
| 0b0010 | Two data value comparators implemented. |
| 0b0011 | Three data value comparators implemented. |
| 0b0100 | Four data value comparators implemented. |
| 0b0101 | Five data value comparators implemented. |
| 0b0110 | Six data value comparators implemented. |
| 0b0111 | Seven data value comparators implemented. |
| 0b1000 | Eight data value comparators implemented. |

All other values are reserved.

This field reads as 0b0000.

NUMACPAIRS, bits [3:0]

Indicates the number of Address Comparator pairs that are available for tracing.

| NUMACPAIRS | Meaning |
|------------|--|
| 0b0000 | No Address Comparator pairs are available. |
| 0b0001 | The implementation has one Address Comparator pair. |
| 0b0010 | The implementation has two Address Comparator pairs. |
| 0b0011 | The implementation has three Address Comparator pairs. |
| 0b0100 | The implementation has four Address Comparator pairs. |
| 0b0101 | The implementation has five Address Comparator pairs. |
| 0b0110 | The implementation has six Address Comparator pairs. |
| 0b0111 | The implementation has seven Address Comparator pairs. |
| 0b1000 | The implementation has eight Address Comparator pairs. |

All other values are reserved.

Accessing TRCIDR4

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR4

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR4;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCIDR4;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR4;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR5, ID Register 5

The TRCIDR5 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR5 bits [31:0] are architecturally mapped to External register [TRCIDR5\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR5 are UNDEFINED.

Attributes

TRCIDR5 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|---------|----|----|----|-------------|----|----|----|------|------------|----|----|----|---------|-------------|----|----|----|------|-------------|----|----|----|----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | NUMCNTR | | | | NUMSEQSTATE | | | | RES0 | LPOVERRIDE | | | | ATBTRIG | TRACEIDSIZE | | | | RES0 | NUMEXTINSEL | | | | NUMEXTIN | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:31]

Reserved, RES0.

NUMCNTR, bits [30:28]

Indicates the number of Counters that are available for tracing.

| NUMCNTR | Meaning |
|---------|-----------------------------|
| 0b000 | No Counters are available. |
| 0b001 | One Counter implemented. |
| 0b010 | Two Counters implemented. |
| 0b011 | Three Counters implemented. |
| 0b100 | Four Counters implemented. |

All other values are reserved.

If [TRCIDR4](#).NUMRSPAIR == 0b0000 then this field is 0b000.

NUMSEQSTATE, bits [27:25]

Indicates if the Sequencer is implemented and the number of Sequencer states that are implemented.

| NUMSEQSTATE | Meaning |
|-------------|--|
| 0b000 | The Sequencer is not implemented. |
| 0b100 | Four Sequencer states are implemented. |

All other values are reserved.

If [TRCIDR4](#).NUMRSPAIR == 0b0000 then this field is 0b000.

Bit [24]

Reserved, RES0.

LPOVERRIDE, bit [23]

Indicates support for Low-power Override Mode.

| LPOVERRIDE | Meaning |
|-------------------|--|
| 0b0 | The trace unit does not support Low-power Override Mode. |
| 0b1 | The trace unit supports Low-power Override Mode. |

ATBTRIG, bit [22]

Indicates if the implementation can support ATB triggers.

| ATBTRIG | Meaning |
|----------------|---|
| 0b0 | The implementation does not support ATB triggers. |
| 0b1 | The implementation supports ATB triggers. |

If [TRCIDR4](#).NUMRSPAIR == 0b0000 then this field is 0.

TRACEIDSIZE, bits [21:16]

Indicates the trace ID width.

| TRACEIDSIZE | Meaning |
|--------------------|--|
| 0b000000 | The external trace interface is not implemented. |
| 0b000111 | The implementation supports a 7-bit trace ID. |

All other values are reserved.

Note that AMBA ATB requires a 7-bit trace ID width.

Bits [15:12]

Reserved, RES0.

NUMEXTINSEL, bits [11:9]

Indicates how many External Input Selector resources are implemented.

| NUMEXTINSEL | Meaning |
|--------------------|---|
| 0b000 | No External Input Selector resources are available. |
| 0b001 | 1 External Input Selector resource is available. |
| 0b010 | 2 External Input Selector resources are available. |
| 0b011 | 3 External Input Selector resources are available. |
| 0b100 | 4 External Input Selector resources are available. |

All other values are reserved.

NUMEXTIN, bits [8:0]

Indicates how many External Inputs are implemented.

| NUMEXTIN | Meaning |
|-----------------|------------------------------|
| 0b11111111 | Unified PMU event selection. |

All other values are reserved.

Accessing TRCIDR5

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR5

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1101 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR5;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR5;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR5;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR6, ID Register 6

The TRCIDR6 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR6 bits [31:0] are architecturally mapped to External register [TRCIDR6\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR6 are UNDEFINED.

Attributes

TRCIDR6 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:3]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [2]

Indicates if Realm EL2 is implemented.

| EXLEVEL_RL_EL2 | Meaning |
|----------------|-------------------------------|
| 0b0 | Realm EL2 is not implemented. |
| 0b1 | Realm EL2 is implemented. |

EXLEVEL_RL_EL1, bit [1]

Indicates if Realm EL1 is implemented.

| EXLEVEL_RL_EL1 | Meaning |
|----------------|-------------------------------|
| 0b0 | Realm EL1 is not implemented. |
| 0b1 | Realm EL1 is implemented. |

EXLEVEL_RL_EL0, bit [0]

Indicates if Realm EL0 is implemented.

| EXLEVEL_RL_EL0 | Meaning |
|----------------|-------------------------------|
| 0b0 | Realm EL0 is not implemented. |
| 0b1 | Realm EL0 is implemented. |

Accessing TRCIDR6

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR6

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1110 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR6;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR6;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIDR6;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR7, ID Register 7

The TRCIDR7 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR7 bits [31:0] are architecturally mapped to External register [TRCIDR7\[31:0\]](#).
This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR7 are UNDEFINED.

Attributes

TRCIDR7 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Reserved, RES0.

Accessing TRCIDR7

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR7

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1111 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR7;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCIDR7;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR7;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR8, ID Register 8

The TRCIDR8 characteristics are:

Purpose

Returns the maximum speculation depth of the instruction trace element stream.

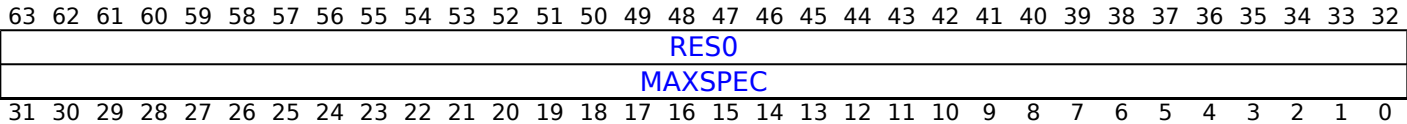
Configuration

AArch64 System register TRCIDR8 bits [31:0] are architecturally mapped to External register [TRCIDR8\[31:0\]](#).
This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR8 are UNDEFINED.

Attributes

TRCIDR8 is a 64-bit register.

Field descriptions



Bits [63:32]

Reserved, RES0.

MAXSPEC, bits [31:0]

Indicates the maximum speculation depth of the instruction trace element stream. This is the maximum number of P0 elements in the trace element stream that can be speculative at any time.

Accessing TRCIDR8

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR8

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR8;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCIDR8;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR8;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR9, ID Register 9

The TRCIDR9 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

AArch64 System register TRCIDR9 bits [31:0] are architecturally mapped to External register [TRCIDR9\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR9 are UNDEFINED.

Attributes

TRCIDR9 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | NUMPOKEY | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

NUMPOKEY, bits [31:0]

When **TRCIDR0.TRCDATA** != 0b00:

Indicates the number of P0 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

Accessing TRCIDR9

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIDR9

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCID == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR9;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCIDR9;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIDR9;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIMSPEC0, IMP DEF Register 0

The TRCIMSPEC0 characteristics are:

Purpose

TRCIMSPEC0 shows the presence of any IMPLEMENTATION DEFINED features, and provides an interface to enable the features that are provided.

Configuration

AArch64 System register TRCIMSPEC0 bits [31:0] are architecturally mapped to External register [TRCIMSPEC0\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIMSPEC0 are UNDEFINED.

Attributes

TRCIMSPEC0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | EN | | | | SUPPORT | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:8]

Reserved, RES0.

EN, bits [7:4]

When TRCIMSPEC0.SUPPORT != 0b0000:

Enable. Controls whether the IMPLEMENTATION DEFINED features are enabled.

| EN | Meaning |
|--------|--|
| 0b0000 | The IMPLEMENTATION DEFINED features are not enabled. The trace unit must behave as if the IMPLEMENTATION DEFINED features are not supported. |
| 0b0001 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b0010 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b0011 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b0100 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b0101 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b0110 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b0111 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1000 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1001 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1010 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1011 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1100 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1101 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1110 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1111 | The trace unit behavior is IMPLEMENTATION DEFINED. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SUPPORT, bits [3:0]

Indicates whether the implementation supports IMPLEMENTATION DEFINED features.

| SUPPORT | Meaning |
|---------|---|
| 0b0000 | No IMPLEMENTATION DEFINED features are supported. |
| 0b0001 | IMPLEMENTATION DEFINED features are supported. |
| 0b0010 | IMPLEMENTATION DEFINED features are supported. |
| 0b0011 | IMPLEMENTATION DEFINED features are supported. |
| 0b0100 | IMPLEMENTATION DEFINED features are supported. |
| 0b0101 | IMPLEMENTATION DEFINED features are supported. |
| 0b0110 | IMPLEMENTATION DEFINED features are supported. |
| 0b0111 | IMPLEMENTATION DEFINED features are supported. |
| 0b1000 | IMPLEMENTATION DEFINED features are supported. |
| 0b1001 | IMPLEMENTATION DEFINED features are supported. |
| 0b1010 | IMPLEMENTATION DEFINED features are supported. |
| 0b1011 | IMPLEMENTATION DEFINED features are supported. |
| 0b1100 | IMPLEMENTATION DEFINED features are supported. |
| 0b1101 | IMPLEMENTATION DEFINED features are supported. |
| 0b1110 | IMPLEMENTATION DEFINED features are supported. |
| 0b1111 | IMPLEMENTATION DEFINED features are supported. |

Use of nonzero values requires written permission from Arm.

Access to this field is **RO**.

Accessing TRCIMSPEC0

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIMSPEC0

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCIMSPEN == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIMSPEC0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIMSPEC0;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIMSPEC0;

```

MSR TRCIMSPEC0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRCIMSPEcn == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCIMSPEC0 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCIMSPEC0 = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCIMSPEC0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIMSPEC<n>, IMP DEF Register <n>, n = 1 - 7

The TRCIMSPEC<n> characteristics are:

Purpose

These registers might return information that is specific to an implementation, or enable features specific to an implementation to be programmed. The product Technical Reference Manual describes these registers.

Configuration

AArch64 System register TRCIMSPEC<n> bits [31:0] are architecturally mapped to External register [TRCIMSPEC<n>\[31:0\]](#).

This register is present only when the trace unit implements this OPTIONAL register and FEAT_ETE is implemented. Otherwise, direct accesses to TRCIMSPEC<n> are UNDEFINED.

Attributes

TRCIMSPEC<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Accessing TRCIMSPEC<n>

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCIMSPEC<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0:n[2:0] | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCIMSPECn == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIMSPEC[UInt(CRm<2:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCIMSPEC[UInt(CRm<2:0>)];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCIMSPEC[UInt(CRm<2:0>)];

```

MSR TRCIMSPEC<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0:n[2:0] | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.TRCIMSPEcn == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCIMSPEC[UInt(CRm<2:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCIMSPEC[UInt(CRm<2:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCIMSPEC[UInt(CRm<2:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCOSLSR, Trace OS Lock Status Register

The TRCOSLSR characteristics are:

Purpose

Returns the status of the Trace OS Lock.

Configuration

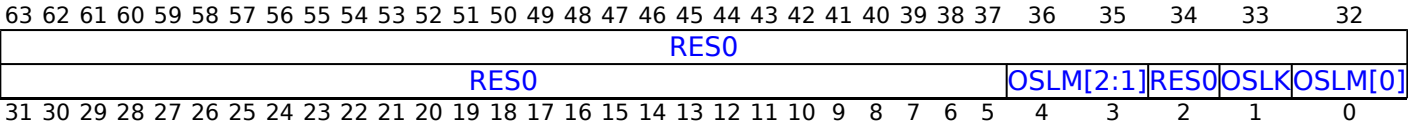
AArch64 System register TRCOSLSR bits [31:0] are architecturally mapped to External register [TRCOSLSR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCOSLSR are UNDEFINED.

Attributes

TRCOSLSR is a 64-bit register.

Field descriptions



Bits [63:5]

Reserved, RES0.

OSLM, bits [4:3, 0]

OS Lock model.

| OSLM | Meaning |
|-------|---|
| 0b000 | Trace OS Lock is not implemented. |
| 0b010 | Trace OS Lock is implemented. |
| 0b100 | Trace OS Lock is not implemented, and the trace unit is controlled by the PE OS Lock. |

All other values are reserved.

This field reads as 0b100.

The OSLM field is split as follows:

- OSLM[2:1] is TRCOSLSR[4:3].
- OSLM[0] is TRCOSLSR[0].

Bit [2]

Reserved, RES0.

OSLK, bit [1]

OS Lock status.

| OSLK | Meaning |
|------|--------------------------|
| 0b0 | The OS Lock is unlocked. |
| 0b1 | The OS Lock is locked. |

Note that this field indicates the state of the PE OS Lock.

Accessing TRCOSLSR

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCOSLSR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0001 | 0b0001 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elseif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elseif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCOSLSR == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCOSLSR;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elseif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elseif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCOSLSR;
elseif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCOSLSR;

```

TRCPRGCTLR, Programming Control Register

The TRCPRGCTLR characteristics are:

Purpose

Enables the trace unit.

Configuration

AArch64 System register TRCPRGCTLR bits [31:0] are architecturally mapped to External register [TRCPRGCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCPRGCTLR are UNDEFINED.

Attributes

TRCPRGCTLR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | EN |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:1]

Reserved, RES0.

EN, bit [0]

Trace unit enable.

| EN | Meaning |
|-----|-----------------------------|
| 0b0 | The trace unit is disabled. |
| 0b1 | The trace unit is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to 0.

Accessing TRCPRGCTLR

Must be programmed.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCPRGCTLR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCPRGCTLR == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCPRGCTLR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCPRGCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCPRGCTLR;

```

MSR TRCPRGCTLR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRCPRGCTLR == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCPRGCTLR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCPRGCTLR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCPRGCTLR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCQCTLR, Q Element Control Register

The TRCQCTLR characteristics are:

Purpose

Controls when Q elements are enabled.

Configuration

AArch64 System register TRCQCTLR bits [31:0] are architecturally mapped to External register [TRCQCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR0.QFILT == 1. Otherwise, direct accesses to TRCQCTLR are UNDEFINED.

Attributes

TRCQCTLR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|------|----------|----------|----------|----------|----------|----------|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | RES0 | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| RES0 | | | | | | | | MODE | RANGE[7] | RANGE[6] | RANGE[5] | RANGE[4] | RANGE[3] | RANGE[2] | | | | | | | | | | | | | | | |

Bits [63:9]

Reserved, RES0.

MODE, bit [8]

Selects whether the Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit is permitted to generate Q elements or address ranges where the trace unit is not permitted to generate Q elements:

| MODE | Meaning |
|------|--|
| 0b0 | Exclude mode. The Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit must not generate Q elements. If no ranges are selected, Q elements are permitted across the entire memory map. |
| 0b1 | Include Mode. The Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit can generate Q elements. If all the implemented bits in RANGE are set to 0 then Q elements are disabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RANGE[<m>], bit [m], for m = 7 to 0

Specifies whether Address Range Comparator <m> controls Q elements.

| RANGE[<m>] | Meaning |
|------------|---|
| 0b0 | The address range that Address Range Comparator <m> defines, is not selected. |
| 0b1 | The address range that Address Range Comparator <m> defines, is selected. |

This bit is RES0 if $m \geq \text{TRCIDR4.NUMACPAIRS}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCQCTLR

Must be programmed if [TRCCONFIGR.QE](#) != 0b00.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCQCTLR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCQCTLR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCQCTLR;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCQCTLR;

```

MSR TRCQCTLR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCQCTLR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCQCTLR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCQCTLR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCRSCTLR<n>, Resource Selection Control Register <n>, n = 2 - 31

The TRCRSCTLR<n> characteristics are:

Purpose

Controls the selection of the resources in the trace unit.

Configuration

AArch64 System register TRCRSCTLR<n> bits [31:0] are architecturally mapped to External register [TRCRSCTLR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and $((\text{TRCIDR4.NUMRSPAIR} + 1) * 2) > n$. Otherwise, direct accesses to TRCRSCTLR<n> are UNDEFINED.

Resource selector 0 always returns FALSE.

Resource selector 1 always returns TRUE.

Resource selectors are implemented in pairs. Each odd numbered resource selector is part of a pair with the even numbered resource selector that is numbered as one less than it. For example, resource selectors 2 and 3 form a pair.

Attributes

TRCRSCTLR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | RES0 | | | | | | | | | | SELECT | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:22]

Reserved, RES0.

PAIRINV, bit [21]

When n is even:

Controls whether the combined result from a resource selector pair is inverted.

| PAIRINV | Meaning |
|---------|--|
| 0b0 | Do not invert the combined output of the 2 resource selectors. |
| 0b1 | Invert the combined output of the 2 resource selectors. |

If:

- A is the register TRCRSCTLR<n>.
- B is the register TRCRSCTLR<n+1>.

Then the combined output of the 2 resource selectors A and B depends on the value of (A.PAIRINV, A.INV, B.INV) as follows:

- 0b000 -> A and B.

- 0b001 -> Reserved.
- 0b010 -> not(A) and B.
- 0b011 -> not(A) and not(B).
- 0b100 -> not(A) or not(B).
- 0b101 -> not(A) or B.
- 0b110 -> Reserved.
- 0b111 -> A or B.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

INV, bit [20]

Controls whether the resource, that TRCRSCTLR<n>.GROUP and TRCRSCTLR<n>.SELECT selects, is inverted.

| INV | Meaning |
|-----|--|
| 0b0 | Do not invert the output of this selector. |
| 0b1 | Invert the output of this selector. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

GROUP, bits [19:16]

Selects a group of resources.

| GROUP | Meaning | SELECT |
|--------|---|--|
| 0b0000 | External Input Selectors. | SELECT encoding for External Input Selectors |
| 0b0001 | PE Comparator Inputs. | SELECT encoding for PE Comparator Inputs |
| 0b0010 | Counters and Sequencer. | SELECT encoding for Counters and Sequencer |
| 0b0011 | Single-shot Comparator Controls. | SELECT encoding for Single-shot Comparator Controls |
| 0b0100 | Single Address Comparators. | SELECT encoding for Single Address Comparators |
| 0b0101 | Address Range Comparators. | SELECT encoding for Address Range Comparators |
| 0b0110 | Context Identifier Comparators. | SELECT encoding for Context Identifier Comparators |
| 0b0111 | Virtual Context Identifier Comparators. | SELECT encoding for Virtual Context Identifier Comparators |

All other values are reserved.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT, bits [15:0]

Resource Specific Controls. Contains the controls specific to the resource group selected by GROUP, described in the following sections.

SELECT encoding for External Input Selectors

| | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|---|---|---|---|---|---|----------|----------|----------|----------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | EXTIN[3] | EXTIN[2] | EXTIN[1] | EXTIN[0] |

Bits [15:4]

Reserved, RES0.

EXTIN[<m>], bit [m], for m = 3 to 0

Selects one or more External Inputs.

| EXTIN[<m>] | Meaning |
|------------|-------------------|
| 0b0 | Ignore EXTIN <m>. |
| 0b1 | Select EXTIN <m>. |

This bit is RES0 if m >= [TRCIDR5](#).NUMEXTINSEL.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for PE Comparator Inputs

| | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|---|---|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | PECOMP[7] | PECOMP[6] | PECOMP[5] | PECOMP[4] | PECOMP[3] | PECOMP[2] | PECOMP[1] | PECOMP[0] |

Bits [15:8]

Reserved, RES0.

PECOMP[<m>], bit [m], for m = 7 to 0

Selects one or more PE Comparator Inputs.

| PECOMP[<m>] | Meaning |
|-------------|---------------------------------|
| 0b0 | Ignore PE Comparator Input <m>. |
| 0b1 | Select PE Comparator Input <m>. |

This bit is RES0 if m >= [TRCIDR4](#).NUMPC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Counters and Sequencer

| | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|---|---|--------------|--------------|--------------|--------------|-------------|-------------|-------------|-------------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | | |
| RES0 | | | | | | | | SEQUENCER[3] | SEQUENCER[2] | SEQUENCER[1] | SEQUENCER[0] | COUNTERS[3] | COUNTERS[2] | COUNTERS[1] | COUNTERS[0] |

Bits [15:8]

Reserved, RES0.

SEQUENCER[<m>], bit [m+4], for m = 3 to 0

Sequencer states.

| SEQUENCER[<m>] | Meaning |
|----------------|-----------------------------|
| 0b0 | Ignore Sequencer state <m>. |
| 0b1 | Select Sequencer state <m>. |

This bit is RES0 if m >= [TRCIDR5](#).NUMSEQSTATE.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

COUNTERS[<m>], bit [m], for m = 3 to 0

Counters resources at zero.

| COUNTERS[<m>] | Meaning |
|---------------|-----------------------------|
| 0b0 | Ignore Counter <m>. |
| 0b1 | Select Counter <m> is zero. |

This bit is RES0 if m >= [TRCIDR5](#).NUMCNTR.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Single-shot Comparator Controls

| | | | | | | | | | | | | | | | |
|------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | SINGLE_SHOT[7] | SINGLE_SHOT[6] | SINGLE_SHOT[5] | SINGLE_SHOT[4] | SINGLE_SHOT[3] | SINGLE_SHOT[2] | SINGLE_SHOT[1] | SINGLE_SHOT[0] | SINGLE_SHOT[7] | SINGLE_SHOT[6] | SINGLE_SHOT[5] | SINGLE_SHOT[4] | SINGLE_SHOT[3] | SINGLE_SHOT[2] | SINGLE_SHOT[1] |

Bits [15:8]

Reserved, RES0.

SINGLE_SHOT[<m>], bit [m], for m = 7 to 0

Selects one or more Single-shot Comparator Controls.

| SINGLE_SHOT[<m>] | Meaning |
|------------------|--|
| 0b0 | Ignore Single-shot Comparator Control <m>. |
| 0b1 | Select Single-shot Comparator Control <m>. |

This bit is RES0 if m >= [TRCIDR4](#).NUMSSCC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Single Address Comparators

| | | | | | | | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SAC[15] | SAC[14] | SAC[13] | SAC[12] | SAC[11] | SAC[10] | SAC[9] | SAC[8] | SAC[7] | SAC[6] | SAC[5] | SAC[4] | SAC[3] | SAC[2] | SAC[1] | SAC[0] |

SAC[<m>], bit [m], for m = 15 to 0

Selects one or more Single Address Comparators.

| SAC[<m>] | Meaning |
|----------|---------------------------------------|
| 0b0 | Ignore Single Address Comparator <m>. |
| 0b1 | Select Single Address Comparator <m>. |

This bit is RES0 if m >= 2 × [TRCIDR4](#).NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Address Range Comparators

| | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|---|---|--------|--------|--------|--------|--------|--------|--------|--------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | ARC[7] | ARC[6] | ARC[5] | ARC[4] | ARC[3] | ARC[2] | ARC[1] | ARC[0] |

Bits [15:8]

Reserved, RES0.

ARC[<m>], bit [m], for m = 7 to 0

Selects one or more Address Range Comparators.

| ARC[<m>] | Meaning |
|----------|--------------------------------------|
| 0b0 | Ignore Address Range Comparator <m>. |
| 0b1 | Select Address Range Comparator <m>. |

This bit is RES0 if m >= [TRCIDR4](#).NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Context Identifier Comparators

| | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|---|---|--------|--------|--------|--------|--------|--------|--------|--------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | CID[7] | CID[6] | CID[5] | CID[4] | CID[3] | CID[2] | CID[1] | CID[0] |

Bits [15:8]

Reserved, RES0.

CID[<m>], bit [m], for m = 7 to 0

Selects one or more Context Identifier Comparators.

| CID[<m>] | Meaning |
|----------|---|
| 0b0 | Ignore Context Identifier Comparator <m>. |
| 0b1 | Select Context Identifier Comparator <m>. |

This bit is RES0 if m >= [TRCIDR4](#).NUMCIDC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Virtual Context Identifier Comparators

| | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|---|---|---------|---------|---------|---------|---------|---------|---------|---------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | VMID[7] | VMID[6] | VMID[5] | VMID[4] | VMID[3] | VMID[2] | VMID[1] | VMID[0] |

Bits [15:8]

Reserved, RES0.

VMID[<m>], bit [m], for m = 7 to 0

Selects one or more Virtual Context Identifier Comparators.

| VMID[<m>] | Meaning |
|-----------|---|
| 0b0 | Ignore Virtual Context Identifier Comparator <m>. |
| 0b1 | Select Virtual Context Identifier Comparator <m>. |

This bit is RES0 if m >= [TRCIDR4.NUMVMIDC](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCRSCTLR<n>

Must be programmed if any of the following are true:

- [TRCCNTCTLR<a>.RLDEVENT](#).TYPE == 0 and [TRCCNTCTLR<a>.RLDEVENT](#).SEL == n.
- [TRCCNTCTLR<a>.RLDEVENT](#).TYPE == 1 and [TRCCNTCTLR<a>.RLDEVENT](#).SEL == n/2.
- [TRCCNTCTLR<a>.CNTEVENT](#).TYPE == 0 and [TRCCNTCTLR<a>.CNTEVENT](#).SEL == n.
- [TRCCNTCTLR<a>.CNTEVENT](#).TYPE == 1 and [TRCCNTCTLR<a>.CNTEVENT](#).SEL == n/2.
- [TRCEVENTCTL0R.EVENT0](#).TYPE == 0 and [TRCEVENTCTL0R.EVENT0](#).SEL == n.
- [TRCEVENTCTL0R.EVENT0](#).TYPE == 1 and [TRCEVENTCTL0R.EVENT0](#).SEL == n/2.
- [TRCEVENTCTL0R.EVENT1](#).TYPE == 0 and [TRCEVENTCTL0R.EVENT1](#).SEL == n.
- [TRCEVENTCTL0R.EVENT1](#).TYPE == 1 and [TRCEVENTCTL0R.EVENT1](#).SEL == n/2.
- [TRCEVENTCTL0R.EVENT2](#).TYPE == 0 and [TRCEVENTCTL0R.EVENT2](#).SEL == n.
- [TRCEVENTCTL0R.EVENT2](#).TYPE == 1 and [TRCEVENTCTL0R.EVENT2](#).SEL == n/2.
- [TRCEVENTCTL0R.EVENT3](#).TYPE == 0 and [TRCEVENTCTL0R.EVENT3](#).SEL == n.
- [TRCEVENTCTL0R.EVENT3](#).TYPE == 1 and [TRCEVENTCTL0R.EVENT3](#).SEL == n/2.
- [TRCSEQEVR<a>.B](#).TYPE == 0 and [TRCSEQEVR<a>.B](#).SEL == n.
- [TRCSEQEVR<a>.B](#).TYPE == 1 and [TRCSEQEVR<a>.B](#).SEL == n/2.
- [TRCSEQEVR<a>.F](#).TYPE == 0 and [TRCSEQEVR<a>.F](#).SEL == n.
- [TRCSEQEVR<a>.F](#).TYPE == 1 and [TRCSEQEVR<a>.F](#).SEL == n/2.
- [TRCSEQRSTEV.RST](#).TYPE == 0 and [TRCSEQRSTEV.RST](#).SEL == n.
- [TRCSEQRSTEV.RST](#).TYPE == 1 and [TRCSEQRSTEV.RST](#).SEL == n/2.
- [TRCTSCTLR.EVENT](#).TYPE == 0 and [TRCTSCTLR.EVENT](#).SEL == n.
- [TRCTSCTLR.EVENT](#).TYPE == 1 and [TRCTSCTLR.EVENT](#).SEL == n/2.
- [TRCVICTLR.EVENT](#).TYPE == 0 and [TRCVICTLR.EVENT](#).SEL == n.
- [TRCVICTLR.EVENT](#).TYPE == 1 and [TRCVICTLR.EVENT](#).SEL == n/2.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCRSCTLR<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-----------|
| 0b10 | 0b001 | 0b0001 | n[3:0] | 0b00:n[4] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCRSCTLR[UInt(op2<0>:CRm<3:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCRSCTLR[UInt(op2<0>:CRm<3:0>)];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCRSCTLR[UInt(op2<0>:CRm<3:0>)];

```

MSR TRCRSCTLR<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-----------|
| 0b10 | 0b001 | 0b0001 | n[3:0] | 0b00:n[4] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCRSCTLR[UInt(op2<0>:CRm<3:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCRSCTLR[UInt(op2<0>:CRm<3:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCRSCTLR[UInt(op2<0>:CRm<3:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCRSR, Resources Status Register

The TRCRSR characteristics are:

Purpose

Use this to set, or read, the status of the resources.

Configuration

AArch64 System register TRCRSR bits [31:0] are architecturally mapped to External register [TRCRSR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCRSR are UNDEFINED.

Attributes

TRCRSR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----------|----------|----------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 |
| RES0 | | | | | | | | | | | | RES0 | | | | | | | | | | | | RES0 | | | | EXTIN[3] | EXTIN[2] | EXTIN[1] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |

Bits [63:13]

Reserved, RES0.

TA, bit [12]

Tracing active.

| TA | Meaning |
|-----|------------------------|
| 0b0 | Tracing is not active. |
| 0b1 | Tracing is active. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

EVENT[<m>], bit [m+8], for m = 3 to 0

Untraced status of ETEEvents.

| EVENT[<m>] | Meaning |
|------------|--|
| 0b0 | An ETEEvent <m> has not occurred. |
| 0b1 | An ETEEvent <m> has occurred while the resources were in the Paused state. |

This bit is RES0 if [TRCIDR4](#).NUMRSPAIR == 0 || m > [TRCIDR0](#).NUMEVENT.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

EXTIN[<m>], bit [m], for m = 3 to 0

The sticky status of the External Input Selectors.

| EXTIN[<m>] | Meaning |
|------------|---|
| 0b0 | An event selected by External Input Selector <m> has not occurred. |
| 0b1 | At least one event selected by External Input Selector <m> has occurred while the resources were in the Paused state. |

This bit is RES0 if m >= [TRCIDR5.NUMEXTINSEL](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCRSR

Must always be programmed.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCRSR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCRSR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCRSR;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCRSR;

```

MSR TRCRSR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCRSR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCRSR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCRSR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSEQEVR<n>, Sequencer State Transition Control Register <n>, n = 0 - 2

The TRCSEQEVR<n> characteristics are:

Purpose

Moves the Sequencer state:

- Backwards, from state n+1 to state n when a programmed resource event occurs.
- Forwards, from state n to state n+1 when a programmed resource event occurs.

Configuration

AArch64 System register TRCSEQEVR<n> bits [31:0] are architecturally mapped to External register [TRCSEQEVR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQEVR<n> are UNDEFINED.

Attributes

TRCSEQEVR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|------|-------|----|----|----|--------|------|-------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | B_TYPE | RES0 | B_SEL | | | | F_TYPE | RES0 | F_SEL | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

B_TYPE, bit [15]

Chooses the type of Resource Selector.

Backward field. Defines whether the backward resource event is a single Resource Selector or a Resource Selector pair. When the resource event occurs then the Sequencer state moves from state n+1 to state n. For example, if TRCSEQEVR2.B.SEL == 0x14 then when event 0x14 occurs, the Sequencer moves from state 3 to state 2.

| B TYPE | Meaning |
|--------|---|
| 0b0 | A single Resource Selector. TRCSEQEVR<n>.B.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCSEQEVR<n>.B.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR<n>.B.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [14:13]

Reserved, RES0.

B_SEL, bits [12:8]

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR<n>.B.TYPE controls whether TRCSEQEVR<n>.B.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Backward field. Selects the single Resource Selector or Resource Selector pair.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

F_TYPE, bit [7]

Chooses the type of Resource Selector.

Backward field. Defines whether the forward resource event is a single Resource Selector or a Resource Selector pair. When the resource event occurs then the Sequencer state moves from state n to state n+1. For example, if TRCSEQEVR1.F.SEL == 0x12 then when event 0x12 occurs, the Sequencer moves from state 1 to state 2.

| F_TYPE | Meaning |
|--------|---|
| 0b0 | A single Resource Selector. TRCSEQEVR<n>.F.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCSEQEVR<n>.F.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR<n>.F.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

F_SEL, bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR<n>.F.TYPE controls whether TRCSEQEVR<n>.F.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Forward field. Selects the single Resource Selector or Resource Selector pair.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQEVR<n>

Must be programmed if [TRCRSCTLR<a>.GROUP == 0b0010](#) and [TRCRSCTLR<a>.SEQUENCER != 0b0000](#).

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCSEQEVR<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|-------|
| 0b10 | 0b001 | 0b0000 | 0b00:n[1:0] | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSEQEVR[UInt(CRm<1:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSEQEVR[UInt(CRm<1:0>)];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSEQEVR[UInt(CRm<1:0>)];

```

MSR TRCSEQEVR<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|-------------|-------|
| 0b10 | 0b001 | 0b0000 | 0b00:n[1:0] | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSEQEVR[UInt(CRm<1:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSEQEVR[UInt(CRm<1:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSEQEVR[UInt(CRm<1:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSEQRSTEVR, Sequencer Reset Control Register

The TRCSEQRSTEVR characteristics are:

Purpose

Moves the Sequencer to state 0 when a programmed resource event occurs.

Configuration

AArch64 System register TRCSEQRSTEVR bits [31:0] are architecturally mapped to External register [TRCSEQRSTEVR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQRSTEVR are UNDEFINED.

Attributes

TRCSEQRSTEVR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:8]

Reserved, RES0.

RST_TYPE, bit [7]

Chooses the type of Resource Selector.

| RST_TYPE | Meaning |
|----------|--|
| 0b0 | A single Resource Selector. TRCSEQRSTEVR.RST.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCSEQRSTEVR.RST.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQRSTEVR.RST.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

RST_SEL, bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQRSTEV.RST.TYPE controls whether TRCSEQRSTEV.RST.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQRSTEV

Must be programmed if [TRCRSCTLR<a>.GROUP == 0b0010](#) and [TRCRSCTLR<a>.SEQUENCER != 0b0000](#).

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCSEQRSTEV

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0110 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSEQRSTEV;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSEQRSTEV;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSEQRSTEV;

```

MSR TRCSEQRSTEV, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0110 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSEQRSTEV = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSEQRSTEV = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSEQRSTEV = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSEQSTR, Sequencer State Register

The TRCSEQSTR characteristics are:

Purpose

Use this to set, or read, the Sequencer state.

Configuration

AArch64 System register TRCSEQSTR bits [31:0] are architecturally mapped to External register [TRCSEQSTR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQSTR are UNDEFINED.

Attributes

TRCSEQSTR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | STATE |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:2]

Reserved, RES0.

STATE, bits [1:0]

Set or returns the state of the Sequencer.

| STATE | Meaning |
|-------|----------|
| 0b00 | State 0. |
| 0b01 | State 1. |
| 0b10 | State 2. |
| 0b11 | State 3. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQSTR

Must be programmed if [TRCRSCTLR<a>](#).GROUP == 0b0010 and [TRCRSCTLR<a>](#).SEQUENCER != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCSEQSTR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0111 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCSEQSTR == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSEQSTR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSEQSTR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSEQSTR;

```

MSR TRCSEQSTR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0111 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRCSEQSTR == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSEQSTR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSEQSTR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSEQSTR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSSCCR<n>, Single-shot Comparator Control Register <n>, n = 0 - 7

The TRCSSCCR<n> characteristics are:

Purpose

Controls the corresponding Single-shot Comparator Control resource.

Configuration

AArch64 System register TRCSSCCR<n> bits [31:0] are architecturally mapped to External register [TRCSSCCR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMSSCC > n. Otherwise, direct accesses to TRCSSCCR<n> are UNDEFINED.

Attributes

TRCSSCCR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|-----|--------|--------|--------|--------|--------|--------|--------|--------|---------|---------|---------|---------|---------|---------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | | | | | | | |
| | | | | | | | | | | | | | | | | | | | RES0 | | | | | | | | |
| RES0 | | | RST | ARC[7] | ARC[6] | ARC[5] | ARC[4] | ARC[3] | ARC[2] | ARC[1] | ARC[0] | SAC[15] | SAC[14] | SAC[13] | SAC[12] | SAC[11] | SAC[10] | SAC[9] | SAC[8] | SAC[7] | SAC[6] | SAC[5] | SAC[4] | SAC[3] | SAC[2] | SAC[1] | SAC[0] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 |

Bits [63:25]

Reserved, RES0.

RST, bit [24]

Selects the Single-shot Comparator Control mode.

| RST | Meaning |
|-----|--|
| 0b0 | The Single-shot Comparator Control is in single-shot mode. |
| 0b1 | The Single-shot Comparator Control is in multi-shot mode. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

ARC[<m>], bit [m+16], for m = 7 to 0

Selects one or more Address Range Comparators for Single-shot control.

| ARC[<m>] | Meaning |
|----------|--|
| 0b0 | The Address Range Comparator <m>, is not selected for Single-shot control. |
| 0b1 | The Address Range Comparator <m>, is selected for Single-shot control. |

This bit is RES0 if m >= [TRCIDR4.NUMACPAIRS](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SAC[<m>], bit [m], for m = 15 to 0

Selects one or more Single Address Comparators for Single-shot control.

| SAC[<m>] | Meaning |
|----------|---|
| 0b0 | The Single Address Comparator <m>, is not selected for Single-shot control. |
| 0b1 | The Single Address Comparator <m>, is selected for Single-shot control. |

This bit is RES0 if $m \geq 2 \times \text{TRCIDR4.NUMACPAIRS}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSSCCR<n>

Must be programmed if any [TRCRSCTLR<a>](#).GROUP == 0b0011 and [TRCRSCTLR<a>](#).SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCSSCCR<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-------|
| 0b10 | 0b001 | 0b0001 | 0b0:n[2:0] | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSSCCR[UInt(CRm<2:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSSCCR[UInt(CRm<2:0>)];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSSCCR[UInt(CRm<2:0>)];

```

MSR TRCSSCCR<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-------|
| 0b10 | 0b001 | 0b0001 | 0b0:n[2:0] | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSSCCR[UInt(CRm<2:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSSCCR[UInt(CRm<2:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSSCCR[UInt(CRm<2:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSSCSR<n>, Single-shot Comparator Control Status Register <n>, n = 0 - 7

The TRCSSCSR<n> characteristics are:

Purpose

Returns the status of the corresponding Single-shot Comparator Control.

Configuration

AArch64 System register TRCSSCSR<n> bits [31:0] are architecturally mapped to External register [TRCSSCSR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMSSCC > n. Otherwise, direct accesses to TRCSSCSR<n> are UNDEFINED.

Attributes

TRCSSCSR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|---------|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|--|----|--|------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| STATUS | | PENDING | | RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | PCDV | | DA | | INST | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | |

Bits [63:32]

Reserved, RES0.

STATUS, bit [31]

Single-shot Comparator Control status. Indicates if any of the comparators selected by this Single-shot Comparator control have matched. The selected comparators are defined by [TRCSSCCR<n>.ARC](#), [TRCSSCCR<n>.SAC](#), and [TRCSSPCICR<n>.PC](#).

| STATUS | Meaning |
|--------|---|
| 0b0 | No match has occurred. When the first match occurs, this field takes a value of 1. It remains at 1 until explicitly modified by a write to this register. |
| 0b1 | One or more matches has occurred. If TRCSSCCR<n>.RST == 0 then: <ul style="list-style-type: none"> There is only one match and no more matches are possible. Software must reset this field to 0 to re-enable the Single-shot Comparator Control. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

PENDING, bit [30]

Single-shot pending status. The Single-shot Comparator Control fired while the resources were in the Paused state.

| PENDING | Meaning |
|---------|-----------------------------------|
| 0b0 | No match has occurred. |
| 0b1 | One or more matches has occurred. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [29:4]

Reserved, RES0.

PC, bit [3]

PE Comparator Input support. Indicates if the Single-shot Comparator Control supports PE Comparator Inputs.

| PC | Meaning |
|-----|--|
| 0b0 | This Single-shot Comparator Control does not support PE Comparator Inputs. Selecting any PE Comparator Inputs using the associated TRCSSPCICR<n> results in CONSTRAINED UNPREDICTABLE behavior of the Single-shot Comparator Control resource. The Single-shot Comparator Control might match unexpectedly or might not match. |
| 0b1 | This Single-shot Comparator Control supports PE Comparator Inputs. |

Access to this field is **RO**.

DV, bit [2]

Data value comparator support. Data value comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

| DV | Meaning |
|-----|--|
| 0b0 | This Single-shot Comparator Control does not support data value comparisons. |
| 0b1 | This Single-shot Comparator Control supports data value comparisons. |

This field reads as 0.

Access to this field is **RO**.

DA, bit [1]

Data Address Comparator support. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

| DA | Meaning |
|-----|--|
| 0b0 | This Single-shot Comparator Control does not support data address comparisons. |
| 0b1 | This Single-shot Comparator Control supports data address comparisons. |

This field reads as 0.

Access to this field is **RO**.

INST, bit [0]

Instruction Address Comparator support. Indicates if the Single-shot Comparator Control supports instruction address comparisons.

| INST | Meaning |
|------|---|
| 0b0 | This Single-shot Comparator Control does not support instruction address comparisons. |
| 0b1 | This Single-shot Comparator Control supports instruction address comparisons. |

This field reads as 1.

Access to this field is **RO**.

Accessing TRCSSCSR<n>

Must be programmed if [TRCRSCTLR<a>](#).GROUP == 0b0011 and [TRCRSCTLR<a>](#).SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCSSCSR<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-------|
| 0b10 | 0b001 | 0b0001 | 0b1:n[2:0] | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCSSCSRn == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSSCSR[UInt(CRm<2:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSSCSR[UInt(CRm<2:0>)];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSSCSR[UInt(CRm<2:0>)];

```

MSR TRCSSCSR<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-------|
| 0b10 | 0b001 | 0b0001 | 0b1.n[2:0] | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRCSSCSRn == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSSCSR[UInt(CRm<2:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSSCSR[UInt(CRm<2:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSSCSR[UInt(CRm<2:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSSPCICR<n>, Single-shot Processing Element Comparator Input Control Register <n>, n = 0 - 7

The TRCSSPCICR<n> characteristics are:

Purpose

Returns the status of the corresponding Single-shot Comparator Control.

Configuration

AArch64 System register TRCSSPCICR<n> bits [31:0] are architecturally mapped to External register [TRCSSPCICR<n>\[31:0\]](#).
This register is present only when FEAT_ETE is implemented, TRCIDR4.NUMSSCC > n, TRCIDR4.NUMPC > 0b0000 and TRCSSCSR<n>.PC == 1. Otherwise, direct accesses to TRCSSPCICR<n> are UNDEFINED.

Attributes

TRCSSPCICR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|-------|----|-------|----|-------|----|-------|----|-------|----|-------|----|-------|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | PC[7] | | PC[6] | | PC[5] | | PC[4] | | PC[3] | | PC[2] | | PC[1] | | PC[0] | |

Bits [63:8]

Reserved, RES0.

PC[<m>], bit [m], for m = 7 to 0

Selects one or more PE Comparator Inputs for Single-shot control.

| PC[<m>] | Meaning |
|---------|---|
| 0b0 | The single PE Comparator Input <m>, is not selected as for Single-shot control. |
| 0b1 | The single PE Comparator Input <m>, is selected as for Single-shot control. |

This bit is RES0 if m >= [TRCIDR4.NUMPC](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSSPCICR<n>

Must be programmed if implemented and any [TRCRSCTLR<a>](#).GROUP == 0b0011 and [TRCRSCTLR<a>](#).SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCSSPCICR<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-------|
| 0b10 | 0b001 | 0b0001 | 0b0:n[2:0] | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSSPCICR[UInt(CRm<2:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSSPCICR[UInt(CRm<2:0>)];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSSPCICR[UInt(CRm<2:0>)];

```

MSR TRCSSPCICR<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-------|
| 0b10 | 0b001 | 0b0001 | 0b0:n[2:0] | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSSPCICR[UInt(CRm<2:0>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSSPCICR[UInt(CRm<2:0>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSSPCICR[UInt(CRm<2:0>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSTALLCTLR, Stall Control Register

The TRCSTALLCTLR characteristics are:

Purpose

Enables trace unit functionality that prevents trace unit buffer overflows.

Configuration

AArch64 System register TRCSTALLCTLR bits [31:0] are architecturally mapped to External register [TRCSTALLCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR3.STALLCTL == 1. Otherwise, direct accesses to TRCSTALLCTLR are UNDEFINED.

Attributes

TRCSTALLCTLR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|------|----|--|----|--------|----|------|--|----|--|-------|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | | 45 | | 44 | 43 | 42 | 41 | | 40 | | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | NOOVERFLOW | | | | RES0 | | | | ISTALL | | RES0 | | | | LEVEL | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | | 13 | | 12 | 11 | 10 | 9 | | 8 | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:14]

Reserved, RES0.

NOOVERFLOW, bit [13]

When TRCIDR3.NOOVERFLOW == 1:

Trace overflow prevention.

| NOOVERFLOW | Meaning |
|------------|--|
| 0b0 | Trace unit buffer overflow prevention is disabled. |
| 0b1 | Trace unit buffer overflow prevention is enabled. |

Note that enabling this feature might cause a significant performance impact.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [12:9]

Reserved, RES0.

ISTALL, bit [8]

Instruction stall control. Controls if a trace unit can stall the PE when the trace buffer space is less than LEVEL.

| ISTALL | Meaning |
|--------|---------------------------------------|
| 0b0 | The trace unit must not stall the PE. |
| 0b1 | The trace unit can stall the PE. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

LEVEL, bits [3:0]

Threshold level field. The field can support 16 monotonic levels from 0b0000 to 0b1111.

The value 0b0000 defines the Minimal invasion level. This setting has a greater risk of a trace unit buffer overflow.

The value 0b1111 defines the Maximum invasion level. This setting has a reduced risk of a trace unit buffer overflow.

Note that for some implementations, invasion might occur at the minimal invasion level.

One or more of the least significant bits of LEVEL are permitted to be RES0. Arm recommends that LEVEL[3:2] are fully implemented. Arm strongly recommends that LEVEL[3] is always implemented. If one or more bits are RES0 and are written with a non-zero value, the effective value of LEVEL is rounded down to the nearest power of 2 value which has the RES0 bits as zero. For example, if LEVEL[1:0] are RES0 and a value of 0b1110 is written to LEVEL, the effective value of LEVEL is 0b1100.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSTALLCTLR

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCSTALLCTLR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSTALLCTLR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCSTALLCTLR;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSTALLCTLR;

```

MSR TRCSTALLCTLR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSTALLCTLR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSTALLCTLR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSTALLCTLR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSTATR, Trace Status Register

The TRCSTATR characteristics are:

Purpose

Returns the trace unit status.

Configuration

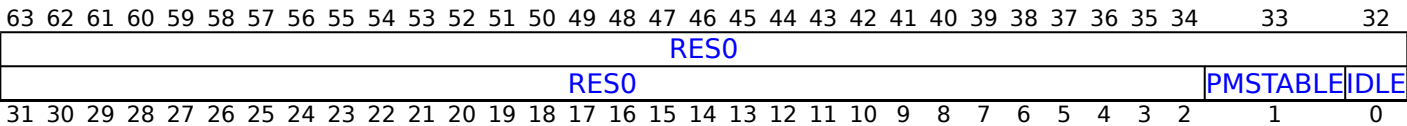
AArch64 System register TRCSTATR bits [31:0] are architecturally mapped to External register [TRCSTATR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCSTATR are UNDEFINED.

Attributes

TRCSTATR is a 64-bit register.

Field descriptions



Bits [63:2]

Reserved, RES0.

PMSTABLE, bit [1]

Programmers' model stable.

| PMSTABLE | Meaning |
|----------|---------------------------------------|
| 0b0 | The programmers' model is not stable. |
| 0b1 | The programmers' model is stable. |

Accessing this field has the following behavior:

- When the trace unit is enabled, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

IDLE, bit [0]

Idle status.

| IDLE | Meaning |
|------|-----------------------------|
| 0b0 | The trace unit is not idle. |
| 0b1 | The trace unit is idle. |

Accessing TRCSTATR

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCSTATR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCSTATR == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSTATR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCSTATR;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSTATR;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSYNCPR, Synchronization Period Register

The TRCSYNCPR characteristics are:

Purpose

Controls how often trace protocol synchronization requests occur.

Configuration

AArch64 System register TRCSYNCPR bits [31:0] are architecturally mapped to External register [TRCSYNCPR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCSYNCPR are UNDEFINED.

Attributes

TRCSYNCPR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | PERIOD | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:5]

Reserved, RES0.

PERIOD, bits [4:0]

Defines the number of bytes of trace between each periodic trace protocol synchronization request.

| PERIOD | Meaning |
|---------|--|
| 0b00000 | Trace protocol synchronization is disabled. |
| 0b01000 | Trace protocol synchronization request occurs after 2^8 bytes of trace. |
| 0b01001 | Trace protocol synchronization request occurs after 2^9 bytes of trace. |
| 0b01010 | Trace protocol synchronization request occurs after 2^{10} bytes of trace. |
| 0b01011 | Trace protocol synchronization request occurs after 2^{11} bytes of trace. |
| 0b01100 | Trace protocol synchronization request occurs after 2^{12} bytes of trace. |
| 0b01101 | Trace protocol synchronization request occurs after 2^{13} bytes of trace. |
| 0b01110 | Trace protocol synchronization request occurs after 2^{14} bytes of trace. |
| 0b01111 | Trace protocol synchronization request occurs after 2^{15} bytes of trace. |
| 0b10000 | Trace protocol synchronization request occurs after 2^{16} bytes of trace. |
| 0b10001 | Trace protocol synchronization request occurs after 2^{17} bytes of trace. |
| 0b10010 | Trace protocol synchronization request occurs after 2^{18} bytes of trace. |
| 0b10011 | Trace protocol synchronization request occurs after 2^{19} bytes of trace. |
| 0b10100 | Trace protocol synchronization request occurs after 2^{20} bytes of trace. |

Other values are reserved. If a reserved value is programmed into PERIOD, then the behavior of the synchronization period counter is CONSTRAINED UNPREDICTABLE and one of the following behaviors occurs:

- No trace protocol synchronization requests are generated by this counter.
- Trace protocol synchronization requests occur at the specified period.
- Trace protocol synchronization requests occur at some other UNKNOWN period which can vary.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSYNCP

Must be programmed if [TRCIDR3](#).SYNCP == 0.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCSYNCP

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSYNCP;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCSYNCP;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCSYNCP;

```

MSR TRCSYNCP, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSYNCPR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCSYNCPR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCSYNCPR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCTRACEIDR, Trace ID Register

The TRCTRACEIDR characteristics are:

Purpose

Sets the trace ID for instruction trace.

Configuration

AArch64 System register TRCTRACEIDR bits [31:0] are architecturally mapped to External register [TRCTRACEIDR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCTRACEIDR are UNDEFINED.

Attributes

TRCTRACEIDR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | TRACEID | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

Bits [63:7]

Reserved, RES0.

TRACEID, bits [6:0]

Trace ID field. Sets the trace ID value for instruction trace. The width of the field is indicated by the value of [TRCIDR5](#).TRACEIDSIZE. Unimplemented bits are RES0.

If an implementation supports AMBA ATB, then:

- The width of the field is 7 bits.
- Writing a reserved trace ID value does not affect behavior of the trace unit but it might cause UNPREDICTABLE behavior of the trace capture infrastructure.

See the AMBA ATB Protocol Specification for information about which ATID values are reserved.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCTRACEIDR

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCTRACEIDR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCTRACEIDR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCTRACEIDR;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCTRACEIDR;

```

MSR TRCTRACEIDR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCTRAIDR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCTRAIDR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCTRAIDR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCTSCTLR, Timestamp Control Register

The TRCTSCTLR characteristics are:

Purpose

Controls the insertion of global timestamps in the trace stream.

Configuration

AArch64 System register TRCTSCTLR bits [31:0] are architecturally mapped to External register [TRCTSCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR0.TSSIZE != 0b00000. Otherwise, direct accesses to TRCTSCTLR are UNDEFINED.

Attributes

TRCTSCTLR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|----|------------|----|------|----|-----------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | | 39 | | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | EVENT_TYPE | | RES0 | | EVENT_SEL | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | | 7 | | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:8]

Reserved, RES0.

EVENT_TYPE, bit [7] When TRCIDR4.NUMRSPAIR != 0b0000:

Chooses the type of Resource Selector.

| EVENT_TYPE | Meaning |
|------------|--|
| 0b0 | A single Resource Selector. TRCTSCTLR.EVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCTSCTLR.EVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCTSCTLR.EVENT.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EVENT_SEL, bits [4:0]

When TRCIDR4.NUMRSPAIR != 0b0000:

Defines the selected Resource Selector or pair of Resource Selectors. TRCTSCTLR.EVENT.TYPE controls whether TRCTSCTLR.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCTSCTLR

Must be programmed if [TRCCONFIGR](#).TS == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCTSCTLR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCTSCTLR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCTSCTLR;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCTSCTLR;

```

MSR TRCTSCTLR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b1100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCTSCTLR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCTSCTLR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCTSCTLR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVICTLR, ViewInst Main Control Register

The TRCVICTLR characteristics are:

Purpose

Controls instruction trace filtering.

Configuration

AArch64 System register TRCVICTLR bits [31:0] are architecturally mapped to External register [TRCVICTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCVICTLR are UNDEFINED.

Attributes

TRCVICTLR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | |
|----------------------|----|----|----|--------------------------------|----|----|----|--------------------------------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 |
| RES0 | | | | EXLEVEL_RL_EL2 | | | | EXLEVEL_RL_EL1 | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |

Bits [63:27]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [26]

When TRCIDR6.EXLEVEL_RL_EL2 == 1:

Filter instruction trace for EL2 in Realm state.

| EXLEVEL_RL_EL2 | Meaning |
|----------------|--|
| 0b0 | When TRCVICTLR.EXLEVEL_NS_EL2 is 0 the trace unit generates instruction trace for EL2 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL2 is 1 the trace unit does not generate instruction trace for EL2 in Realm state. |
| 0b1 | When TRCVICTLR.EXLEVEL_NS_EL2 is 0 the trace unit does not generate instruction trace for EL2 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL2 is 1 the trace unit generates instruction trace for EL2 in Realm state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL1, bit [25]**When TRCIDR6.EXLEVEL_RL_EL1 == 1:**

Filter instruction trace for EL1 in Realm state.

| EXLEVEL_RL_EL1 | Meaning |
|----------------|--|
| 0b0 | When TRCVICTLR.EXLEVEL_NS_EL1 is 0 the trace unit generates instruction trace for EL1 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL1 is 1 the trace unit does not generate instruction trace for EL1 in Realm state. |
| 0b1 | When TRCVICTLR.EXLEVEL_NS_EL1 is 0 the trace unit does not generate instruction trace for EL1 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL1 is 1 the trace unit generates instruction trace for EL1 in Realm state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_ELO, bit [24]**When TRCIDR6.EXLEVEL_RL_ELO == 1:**

Filter instruction trace for EL0 in Realm state.

| EXLEVEL_RL_ELO | Meaning |
|----------------|--|
| 0b0 | When TRCVICTLR.EXLEVEL_NS_ELO is 0 the trace unit generates instruction trace for EL0 in Realm state. When TRCVICTLR.EXLEVEL_NS_ELO is 1 the trace unit does not generate instruction trace for EL0 in Realm state. |
| 0b1 | When TRCVICTLR.EXLEVEL_NS_ELO is 0 the trace unit does not generate instruction trace for EL0 in Realm state. When TRCVICTLR.EXLEVEL_NS_ELO is 1 the trace unit generates instruction trace for EL0 in Realm state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [22]**When Non-secure EL2 is implemented:**

Filter instruction trace for EL2 in Non-secure state.

| EXLEVEL_NS_EL2 | Meaning |
|----------------|---|
| 0b0 | The trace unit generates instruction trace for EL2 in Non-secure state. |
| 0b1 | The trace unit does not generate instruction trace for EL2 in Non-secure state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL1, bit [21]

When Non-secure EL1 is implemented:

Filter instruction trace for EL1 in Non-secure state.

| EXLEVEL_NS_EL1 | Meaning |
|----------------|---|
| 0b0 | The trace unit generates instruction trace for EL1 in Non-secure state. |
| 0b1 | The trace unit does not generate instruction trace for EL1 in Non-secure state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_ELO, bit [20]

When Non-secure ELO is implemented:

Filter instruction trace for ELO in Non-secure state.

| EXLEVEL_NS_ELO | Meaning |
|----------------|---|
| 0b0 | The trace unit generates instruction trace for ELO in Non-secure state. |
| 0b1 | The trace unit does not generate instruction trace for ELO in Non-secure state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL3, bit [19]

When EL3 is implemented:

Filter instruction trace for EL3.

| EXLEVEL_S_EL3 | Meaning |
|---------------|---|
| 0b0 | The trace unit generates instruction trace for EL3. |
| 0b1 | The trace unit does not generate instruction trace for EL3. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL2, bit [18]

When EL2 is implemented and FEAT_SEL2 is implemented:

Filter instruction trace for EL2 in Secure state.

| EXLEVEL_S_EL2 | Meaning |
|---------------|---|
| 0b0 | The trace unit generates instruction trace for EL2 in Secure state. |
| 0b1 | The trace unit does not generate instruction trace for EL2 in Secure state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL1, bit [17]

When Secure EL1 is implemented:

Filter instruction trace for EL1 in Secure state.

| EXLEVEL_S_EL1 | Meaning |
|---------------|---|
| 0b0 | The trace unit generates instruction trace for EL1 in Secure state. |
| 0b1 | The trace unit does not generate instruction trace for EL1 in Secure state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL0, bit [16]

When Secure EL0 is implemented:

Filter instruction trace for EL0 in Secure state.

| EXLEVEL S_ELO | Meaning |
|---------------|---|
| 0b0 | The trace unit generates instruction trace for ELO in Secure state. |
| 0b1 | The trace unit does not generate instruction trace for ELO in Secure state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [15:12]

Reserved, RES0.

TRCERR, bit [11]

When TRCIDR3.TRCERR == 1:

Controls the forced tracing of System Error exceptions.

| TRCERR | Meaning |
|--------|--|
| 0b0 | Forced tracing of System Error exceptions is disabled. |
| 0b1 | Forced tracing of System Error exceptions is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCRESET, bit [10]

Controls the forced tracing of PE Resets.

| TRCRESET | Meaning |
|----------|--|
| 0b0 | Forced tracing of PE Resets is disabled. |
| 0b1 | Forced tracing of PE Resets is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SSSTATUS, bit [9]

ViewInst start/stop function status.

| SSSTATUS | Meaning |
|----------|---|
| 0b0 | Stopped State. The ViewInst start/stop function is in the stopped state. |
| 0b1 | Started State. The ViewInst start/stop function is in the started state. |

Before software enables the trace unit, it must write to this field to set the initial state of the ViewInst start/stop function. If the ViewInst start/stop function is not used then set this field to 1. Arm recommends that the value of this field is set before each trace session begins.

If the trace unit becomes disabled while a start point or stop point is still speculative, then the value of TRCVICTLR.SSSTATUS is UNKNOWN and might represent the result of a speculative start point or stop point.

If software which is running on the PE being traced disables the trace unit, either by clearing [TRCPRGCTLR.EN](#) or locking the OS Lock, Arm recommends that a DSB and an ISB instruction are executed before disabling the trace unit to prevent any start points or stop points being speculative at the point of disabling the trace unit. This procedure assumes that all start points or stop points occur before the barrier instructions are executed. The procedure does not guarantee that there are no speculative start points or stop points when disabling, although it helps minimize the probability.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- RES1** if all of the following are true:
 - TRCIDR4.NUMACPAIRS == 0b0000
 - TRCIDR4.NUMPC == 0b0000
- Otherwise, access to this field is **RW**.

Bit [8]

Reserved, RES0.

EVENT_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != 0b0000:

Chooses the type of Resource Selector.

| EVENT_TYPE | Meaning |
|------------|--|
| 0b0 | A single Resource Selector. TRCVICTLR.EVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCVICTLR.EVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCVICTLR.EVENT.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EVENT_SEL, bits [4:0]

When TRCIDR4.NUMRSPAIR != 0b0000:

Defines the selected Resource Selector or pair of Resource Selectors. TRCVICTLR.EVENT.TYPE controls whether TRCVICTLR.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

When TRCIDR4.NUMRSPAIR == 0b0000:

This field is reserved:

- Bits [4:1] are RES0.
- Bit [0] is RES1.

Otherwise:

Reserved, RES0.

Accessing TRCVICTLR

Must be programmed.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCVICTLR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRCVICTLR == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCVICTLR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCVICTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCVICTLR;

```

MSR TRCVICTLR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.TRCVICTLR == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVICTLR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVICTLR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVICTLR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVIICTRL, ViewInst Include/Exclude Control Register

The TRCVIICTLR characteristics are:

Purpose

Use this to select, or read, the Address Range Comparators for the ViewInst include/exclude function.

Configuration

AArch64 System register TRCVIIECTLR bits [31:0] are architecturally mapped to External register [TRCVIIECTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMACPAIRS > 0b0000. Otherwise, direct accesses to TRCVIIECTLR are UNDEFINED.

Attributes

TRCVIIECTLR is a 64-bit register.

Field descriptions

| | | | | | | | | | |
|------------------|------------|------------|------------|------------|------------|------------|------------|------------|------|
| 6362616059585756 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 |
| | | | | | | | | | RES0 |
| RES0 | EXCLUDE[7] | EXCLUDE[6] | EXCLUDE[5] | EXCLUDE[4] | EXCLUDE[3] | EXCLUDE[2] | EXCLUDE[1] | EXCLUDE[0] | |
| 3130292827262524 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 |

Bits [63:24]

Reserved, RES0.

EXCLUDE[<m>], bit [m+16], for m = 7 to 0

Exclude Address Range Comparator <m>. Selects whether Address Range Comparator <m> is in use with the ViewInst exclude function.

| EXCLUDE[<m>] | Meaning |
|---------------------------|---|
| 0b0 | The address range that Address Range Comparator <m> defines, is not selected for the ViewInst exclude function. |
| 0b1 | The address range that Address Range Comparator <m> defines, is selected for the ViewInst exclude function. |

This bit is RES0 if $m \geq \text{TRCIDR4.NUMACPAIRS}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [15:8]

Reserved, RES0.

INCLUDE[<m>], bit [m], for m = 7 to 0

Include Address Range Comparator <m>.

Selects whether Address Range Comparator <m> is in use with the ViewInst include function.

Selecting no comparators for the ViewInst include function indicates that all instructions are included by default.

The ViewInst exclude function then indicates which ranges are excluded.

| INCLUDE[<m>] | Meaning |
|---------------------------|---|
| 0b0 | The address range that Address Range Comparator <m> defines, is not selected for the ViewInst include function. |
| 0b1 | The address range that Address Range Comparator <m> defines, is selected for the ViewInst include function. |

This bit is RES0 if m >= [TRCIDR4](#).NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCVIIECTLR

Must be programmed if [TRCIDR4](#).NUMACPAIRS > 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCVIIECTLR

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b10 | 0b001 | 0b0000 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCVIIECTLR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCVIIECTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCVIIECTLR;

```

MSR TRCVIIECTLR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVIIIECTLR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVIIIECTLR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVIIIECTLR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVIPCSSCTLR, ViewInst Start/Stop PE Comparator Control Register

The TRCVIPCSSCTLR characteristics are:

Purpose

Use this to select, or read, which PE Comparator Inputs can control the ViewInst start/stop function.

Configuration

AArch64 System register TRCVIPCSSCTLR bits [31:0] are architecturally mapped to External register [TRCVIPCSSCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMPC > 0b0000. Otherwise, direct accesses to TRCVIPCSSCTLR are UNDEFINED.

Attributes

TRCVIPCSSCTLR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | |
|------------------|---------|---------|---------|---------|---------|---------|---------|---------|------------------|----------|-------|
| 6362616059585756 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 4746454443424140 | 39 | 38 |
| RES0 | | | | | | | | | | | |
| RES0 | STOP[7] | STOP[6] | STOP[5] | STOP[4] | STOP[3] | STOP[2] | STOP[1] | STOP[0] | RES0 | START[7] | START |
| 3130292827262524 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15141312111098 | 7 | 6 |

Bits [63:24]

Reserved, RES0.

STOP[<m>], bit [m+16], for m = 7 to 0

Selects whether PE Comparator Input <m> is in use with ViewInst start/stop function, for the purpose of stopping trace.

| STOP[<m>] | Meaning |
|-----------|--|
| 0b0 | The PE Comparator Input <m>, is not selected as a stop resource. |
| 0b1 | The PE Comparator Input <m>, is selected as a stop resource. |

This bit is RES0 if m >= [TRCIDR4](#).NUMPC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [15:8]

Reserved, RES0.

START[<m>], bit [m], for m = 7 to 0

Selects whether PE Comparator Input <m> is in use with ViewInst start/stop function, for the purpose of starting trace.

| START[<m>] | Meaning |
|------------|---|
| 0b0 | The PE Comparator Input <m>, is not selected as a start resource. |
| 0b1 | The PE Comparator Input <m>, is selected as a start resource. |

This bit is RES0 if m >= [TRCIDR4](#).NUMPC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCVIPCSSCTLR

Must be programmed if [TRCIDR4](#).NUMPC != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCVIPCSSCTLR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCVIPCSSCTLR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCVIPCSSCTLR;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCVIPCSSCTLR;

```

MSR TRCVIPCSSCTLR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVIPCSSCTLR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVIPCSSCTLR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVIPCSSCTLR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVISSCTLR, ViewInst Start/Stop Control Register

The TRCVISSCTLR characteristics are:

Purpose

Use this to select, or read, the Single Address Comparators for the ViewInst start/stop function.

Configuration

AArch64 System register TRCVISSCTLR bits [31:0] are architecturally mapped to External register [TRCVISSCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMACPAIRS > 0b0000. Otherwise, direct accesses to TRCVISSCTLR are UNDEFINED.

Attributes

TRCVISSCTLR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 |
| STOP[15] | STOP[14] | STOP[13] | STOP[12] | STOP[11] | STOP[10] | STOP[9] | STOP[8] | STOP[7] | STOP[6] | STOP[5] | STOP[4] | STOP[3] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 |

Bits [63:32]

Reserved, RES0.

STOP[<m>], bit [m+16], for m = 15 to 0

Selects whether Single Address Comparator <m> is used with the ViewInst start/stop function, for the purpose of stopping trace.

| STOP[<m>] | Meaning |
|-----------|--|
| 0b0 | The Single Address Comparator <m>, is not selected as a stop resource. |
| 0b1 | The Single Address Comparator <m>, is selected as a stop resource. |

This bit is RES0 if $m \geq 2 \times \text{TRCIDR4.NUMACPAIRS}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

START[<m>], bit [m], for m = 15 to 0

Selects whether Single Address Comparator <m> is used with the ViewInst start/stop function, for the purpose of starting trace.

| START[<m>] | Meaning |
|------------|---|
| 0b0 | The Single Address Comparator <m>, is not selected as a start resource. |
| 0b1 | The Single Address Comparator <m>, is selected as a start resource. |

This bit is RES0 if $m \geq 2 \times \text{TRCIDR4.NUMACPAIRS}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCVISSCTLR

Must be programmed if $\text{TRCIDR4.NUMACPAIRS} > 0b0000$.

For any 2 comparators selected for the ViewInst start/stop function, the comparator containing the lower address must be a lower numbered comparator.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCVISSCTLR

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCVISSCTLR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCVISSCTLR;
        elsif PSTATE.EL == EL3 then
            if CPTR_EL3.TTA == '1' then
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCVISSCTLR;

```

MSR TRCVISSCTLR, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0000 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVISSCTLR = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVISSCTLR = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVISSCTLR = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVMIDCCTL0, Virtual Context Identifier Comparator Control Register 0

The TRCVMIDCCTL0 characteristics are:

Purpose

Virtual Context Identifier Comparator mask values for the [TRCVMIDCVR<n>](#) registers, where n=0-3.

Configuration

AArch64 System register TRCVMIDCCCTLRO bits [31:0] are architecturally mapped to External register [TRCVMIDCCCTLRO\[31:0\]](#).

This register is present only when FEAT_ETE is implemented, TRCIDR4.NUMVMIDC > 0x0 and TRCIDR2.VMIDSIZE > 0b00000. Otherwise, direct accesses to TRCVMIDCCTLR0 are UNDEFINED.

Attributes

TRCVMIDCCTL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 |
| | | | | | | | | | | |
| COMP3[7] | COMP3[6] | COMP3[5] | COMP3[4] | COMP3[3] | COMP3[2] | COMP3[1] | COMP3[0] | COMP2[7] | COMP2[6] | COMP2[5] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 |

Bits [63:32]

Reserved, RES0.

COMP3[<m>], bit [m+24], for m = 7 to 0

When TRCIDR4.NUMVMIDC > 3:

TRCVMIDCVR3 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR3. Each bit in this field corresponds to a byte in TRCVMIDCVR3.

| COMP3[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if $m \geq \text{TRCIDR2.VMIDSIZE}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP2[<m>], bit [m+16], for m = 7 to 0**When TRCIDR4.NUMVMIDC > 2:**

TRCVMIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR2. Each bit in this field corresponds to a byte in TRCVMIDCVR2.

| COMP2[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR2[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR2[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.VMIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP1[<m>], bit [m+8], for m = 7 to 0**When TRCIDR4.NUMVMIDC > 1:**

TRCVMIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR1. Each bit in this field corresponds to a byte in TRCVMIDCVR1.

| COMP1[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR1[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR1[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.VMIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP0[<m>], bit [m], for m = 7 to 0**When TRCIDR4.NUMVMIDC > 0:**

TRCVMIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR0. Each bit in this field corresponds to a byte in TRCVMIDCVR0.

| COMP0[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR0[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR0[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.VMIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCVMIDCCTLRO

If software uses the [TRCVMIDCVR<n>](#) registers, where n=0-3, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in [TRCVMIDCVR<n>](#) to 0x00.

If any bit is 1 and the relevant byte in [TRCVMIDCVR<n>](#) is not 0x00, the behavior of the Virtual Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCVMIDCCTLRO

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0011 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCVMIDCCTLRO;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
            UNDEFINED;
        elsif CPTR_EL2.TTA == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
            else
                return TRCVMIDCCTLRO;
    elsif PSTATE.EL == EL3 then
        if CPTR_EL3.TTA == '1' then
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCVMIDCCTLRO;

```

MSR TRCVMIDCCTLR0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0011 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVMIDCCTLR0 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVMIDCCTLR0 = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVMIDCCTLR0 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVMIDCCTL1, Virtual Context Identifier Comparator Control Register 1

The TRCVMIDCCTL1 characteristics are:

Purpose

Virtual Context Identifier Comparator mask values for the [TRCVMIDCVR<n>](#) registers, where n=4-7.

Configuration

AArch64 System register TRCVMIDCCTL1R1 bits [31:0] are architecturally mapped to External register [TRCVMIDCCTL1R1\[31:0\]](#).

This register is present only when FEAT_ETE is implemented, TRCIDR4.NUMVMIDC > 0x4 and TRCIDR2.VMIDSIZE > 0b00000. Otherwise, direct accesses to TRCVMIDCCTLR1 are UNDEFINED.

Attributes

TRCVMIDCCTL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 |
| | | | | | | | | | | |
| COMP7[7] | COMP7[6] | COMP7[5] | COMP7[4] | COMP7[3] | COMP7[2] | COMP7[1] | COMP7[0] | COMP6[7] | COMP6[6] | COMP6[5] |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 |

Bits [63:32]

Reserved, RES0.

COMP7[<m>], bit [m+24], for m = 7 to 0

When TRCIDR4.NUMVMIDC > 7:

TRCVMIDCVR7 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR7. Each bit in this field corresponds to a byte in TRCVMIDCVR7.

| COMP7[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if $m \geq \text{TRCIDR2.VMIDSIZE}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP6[<m>], bit [m+16], for m = 7 to 0**When TRCIDR4.NUMVMIDC > 6:**

TRCVMIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR6. Each bit in this field corresponds to a byte in TRCVMIDCVR6.

| COMP6[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR6[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR6[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.VMIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP5[<m>], bit [m+8], for m = 7 to 0**When TRCIDR4.NUMVMIDC > 5:**

TRCVMIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR5. Each bit in this field corresponds to a byte in TRCVMIDCVR5.

| COMP5[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR5[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR5[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.VMIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP4[<m>], bit [m], for m = 7 to 0**When TRCIDR4.NUMVMIDC > 4:**

TRCVMIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR4. Each bit in this field corresponds to a byte in TRCVMIDCVR4.

| COMP4[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR4[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR4[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.VMIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCVMIDCCTLR1

If software uses the [TRCVMIDCVR<n>](#) registers, where n=4-7, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in [TRCVMIDCVR<n>](#) to 0x00.

If any bit is 1 and the relevant byte in [TRCVMIDCVR<n>](#) is not 0x00, the behavior of the Virtual Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCVMIDCCTLR1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0011 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCVMIDCCTLR1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCVMIDCCTLR1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCVMIDCCTLR1;

```

MSR TRCVMIDCCTLR1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b10 | 0b001 | 0b0011 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVMIDCCTLR1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVMIDCCTLR1 = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVMIDCCTLR1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVMIDCVR<n>, Virtual Context Identifier Comparator Value Register <n>, n = 0 - 7

The TRCVMIDCVR<n> characteristics are:

Purpose

Contains the Virtual Context Identifier Comparator value.

Configuration

AArch64 System register TRCVMIDCVR<n> bits [63:0] are architecturally mapped to External register [TRCVMIDCVR<n>\[63:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMVMIDC > n. Otherwise, direct accesses to TRCVMIDCVR<n> are UNDEFINED.

Attributes

TRCVMIDCVR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

VALUE, bits [63:0]

Virtual context identifier value. The width of this field is indicated by [TRCIDR2.VMIDSIZE](#). Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Virtual context identifier is zero until the PE updates the Virtual context identifier .

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCVMIDCVR<n>

Must be programmed if any of the following are true:

- [TRCRSCTLR<a>](#).GROUP == 0b0111 and [TRCRSCTLR<a>](#).VMID[n] == 1.
- [TRCACATR<a>](#).CONTEXTTYPE == 0b10 or 0b11 and [TRCACATR<a>](#).CONTEXT == n.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRCVMIDCVR<n>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-------|
| 0b10 | 0b001 | 0b0011 | n[2:0]:0b0 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCVMIDCVR[UInt(CRm<3:1>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRCVMIDCVR[UInt(CRm<3:1>)];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRCVMIDCVR[UInt(CRm<3:1>)];

```

MSR TRCVMIDCVR<n>, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|------------|-------|
| 0b10 | 0b001 | 0b0011 | n[2:0]:0b0 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPACR_EL1.TTA == '1' then
        AArch64.SystemAccessTrap(EL1, 0x18);
    elsif EL2Enabled() && CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.TRC == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVMIDCVR[UInt(CRm<3:1>)] = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.TTA == '1' then
        UNDEFINED;
    elsif CPTR_EL2.TTA == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && CPTR_EL3.TTA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRCVMIDCVR[UInt(CRm<3:1>)] = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.TTA == '1' then
        AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRCVMIDCVR[UInt(CRm<3:1>)] = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRFCR_EL1, Trace Filter Control Register (EL1)

The TRFCR_EL1 characteristics are:

Purpose

Provides EL1 controls for Trace.

Configuration

AArch64 System register TRFCR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [TRFCR\[31:0\]](#).

This register is present only when FEAT_TRF is implemented. Otherwise, direct accesses to TRFCR_EL1 are UNDEFINED.

Attributes

TRFCR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|-------|----|-------|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | TS | | RES0 | | E1TRE | | E0TRE | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control. Controls which timebase is used for trace timestamps.

| TS | Meaning | Applies when |
|------|---|------------------------------|
| 0b01 | Virtual timestamp. The traced timestamp is the physical counter value minus the value of CNTVOFF_EL2 . | |
| 0b10 | Guest physical timestamp. The traced timestamp is the physical counter value minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2 : <ul style="list-style-type: none"> • SCR_EL3.ECVEN == 0. • CNTHCTL_EL2.ECV == 0. | When FEAT_ECV is implemented |
| 0b11 | Physical timestamp. The traced timestamp is the physical counter value. | |

All other values are reserved.

This field is ignored by the PE when any of the following are true:

- EL2 is implemented and `TRFCR_EL2.TS != 0b00`.
- `SelfHostedTraceEnabled() == FALSE`.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:2]

Reserved, RES0.

E1TRE, bit [1]

EL1 Trace Enable.

| E1TRE | Meaning |
|--------------|-----------------------------|
| 0b0 | Trace is prohibited at EL1. |
| 0b1 | Trace is allowed at EL1. |

This field is ignored if SelfHostedTraceEnabled() == FALSE.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

E0TRE, bit [0]

EL0 Trace Enable.

| E0TRE | Meaning |
|--------------|-----------------------------|
| 0b0 | Trace is prohibited at EL0. |
| 0b1 | Trace is allowed at EL0. |

This field is ignored if any of the following are true:

- SelfHostedTraceEnabled() == FALSE.
- EL2 is implemented and enabled in the current Security state and [HCR_EL2.TGE](#) == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing TRFCR_EL1

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRFCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------------|------------|------------|------------|------------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x880];
    else
        return TRFCR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return TRFCR_EL2;
    else
        return TRFCR_EL1;
elsif PSTATE.EL == EL3 then
    return TRFCR_EL1;

```

MSR TRFCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.TRFCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x880] = X[t];
    else
        TRFCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        TRFCR_EL2 = X[t];
    else
        TRFCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TRFCR_EL1 = X[t];

```

MRS <Xt>, TRFCR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0001 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x880];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TTRF == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            return TRFCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return TRFCR_EL1;
    else
        UNDEFINED;

```

MSR TRFCR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0001 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x880] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TTRF == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x18);
        else
            TRFCR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        TRFCR_EL1 = X[t];
    else
        UNDEFINED;

```


TRFCR_EL2, Trace Filter Control Register (EL2)

The TRFCR_EL2 characteristics are:

Purpose

Provides EL2 controls for Trace.

Configuration

AArch64 System register TRFCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HTRFCR\[31:0\]](#).

This register is present only when FEAT_TRF is implemented. Otherwise, direct accesses to TRFCR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

TRFCR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|------|----|------|-------|--------|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | TS | | RES0 | CX | RES0 | E2TRE | E0HTRE | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control. Controls which timebase is used for trace timestamps.

| TS | Meaning | Applies when |
|------|---|------------------------------|
| 0b00 | Timestamp controlled by TRFCR_EL1 .TS or TRFCR .TS. | |
| 0b01 | Virtual timestamp. The traced timestamp is the physical counter value minus the value of CNTVOFF_EL2 . | |
| 0b10 | Guest physical timestamp. The traced timestamp is the physical counter value minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2 : <ul style="list-style-type: none"> SCR_EL3.ECVEn == 0. CNTHCTL_EL2.ECV == 0. | When FEAT_ECV is implemented |
| 0b11 | Physical timestamp. The traced timestamp is the physical counter value. | |

This field is ignored by the PE when SelfHostedTraceEnabled() == FALSE.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bit [4]

Reserved, RES0.

CX, bit [3]

[CONTEXTIDR_EL2](#) and VMID trace enable.

| CX | Meaning |
|-----|---|
| 0b0 | CONTEXTIDR_EL2 and VMID trace prohibited. |
| 0b1 | CONTEXTIDR_EL2 and VMID trace allowed. |

This field is ignored if SelfHostedTraceEnabled() == FALSE.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bit [2]

Reserved, RES0.

E2TRE, bit [1]

EL2 Trace Enable.

| E2TRE | Meaning |
|-------|-----------------------------|
| 0b0 | Trace is prohibited at EL2. |
| 0b1 | Trace is allowed at EL2. |

This field is ignored if SelfHostedTraceEnabled() == FALSE.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

EOHTRE, bit [0]

EL0 Trace Enable.

| EOHTRE | Meaning |
|--------|---|
| 0b0 | Trace is prohibited at EL0 when HCR_EL2.TGE == 1. |
| 0b1 | Trace is allowed at EL0 when HCR_EL2.TGE == 1. |

This field is ignored if any of the following are true:

- SelfHostedTraceEnabled() == FALSE.
- EL2 is disabled in the current security state.
- [HCR_EL2.TGE](#) == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing TRFCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TRFCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        return TRFCR_EL2;
elsif PSTATE.EL == EL3 then
    return TRFCR_EL2;

```

MSR TRFCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    else
        TRFCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    TRFCR_EL2 = X[t];

```

MRS <Xt>, TRFCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif EL2Enabled() && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x880];
    else
        return TRFCR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        return TRFCR_EL2;
    else
        return TRFCR_EL1;
elsif PSTATE.EL == EL3 then
    return TRFCR_EL1;

```

MSR TRFCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.TRFCR_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && MDCR_EL2.TTRF == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x880] = X[t];
    else
        TRFCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x18);
    elsif HCR_EL2.E2H == '1' then
        TRFCR_EL2 = X[t];
    else
        TRFCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TRFCR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBR0_EL1, Translation Table Base Register 0 (EL1)

The TTBR0_EL1 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the lower VA range in the EL1&0 translation regime, and other information for this translation regime.

Configuration

AArch64 System register TTBR0_EL1 bits [63:0] are architecturally mapped to AArch32 System register [TTBR0\[63:0\]](#).

Attributes

TTBR0_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | BADDR[47:1] | | | | | | | | | | | | | | | |
| BADDR[47:1] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | CnP |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

An ASID for the translation table base address. The [TCR_EL1.A1](#) field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR[47:1], bits [47:1]

Translation table base address:

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The smallest permitted value of x is 6. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of [TCR_EL1.T0SZ](#), the translation stage, and the translation granule size.

Note

A translation table is required to be aligned to the size of the table. If a table contains fewer than eight entries, it must be aligned on a 64 byte address boundary.

If the value of [TCR_EL1.IPS](#) is not 0b110, then:

- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits A[51:48] of the stage 1 translation table base address are 0b0000.

If FEAT_LPA is implemented and the value of [TCR_EL1](#).IPS is 0b110, then:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
- Register bit[1] is RES0.
- When $x > 6$, register bits[($x-1$):6] are RES0.

Note

[TCR_EL1](#).IPS==0b110 is permitted when:

- FEAT_LPA is implemented and the 64KB translation granule is used.
- FEAT_LPA2 is implemented and the 4KB or 16KB translation granule is used.

When the value of [ID_AA64MMFR0_EL1](#).PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register when the Effective value of [TCR_EL1](#).IPS is 0b110 and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR0_EL1, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[($x-1$):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1.

| CnP | Meaning |
|-----|---|
| 0b0 | <p>The translation table entries pointed to by TTBR0_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR0_EL1 for other PEs in the Inner Shareable domain. This is not affected by:</p> <ul style="list-style-type: none"> • The value of TTBR0_EL1.CnP on those other PEs. • The value of the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID. |
| 0b1 | <p>The translation table entries pointed to by TTBR0_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL1.CnP is 1 and all of the following apply:</p> <ul style="list-style-type: none"> • The translation table entries are pointed to by TTBR0_EL1. • The translation tables relate to the same translation regime. • The ASID is the same as the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID. |

This bit is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR0_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR0_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic TTBR0_EL1 or TTBR0_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TTBR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.TTBR0_EL1 == '1'
    then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x200];
    else
        return TTBR0_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TTBR0_EL2;
    else
        return TTBR0_EL1;
elsif PSTATE.EL == EL3 then
    return TTBR0_EL1;

```

MSR TTBR0_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.TTBR0_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x200] = X[t];
    else
        TTBR0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        TTBR0_EL2 = X[t];
    else
        TTBR0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TTBR0_EL1 = X[t];

```

MRS <Xt>, TTBR0_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0010 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x200];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TTBR0_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return TTBR0_EL1;
    else
        UNDEFINED;

```

MSR TTBR0_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0010 | 0b0000 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x200] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        TTBR0_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        TTBR0_EL1 = X[t];
    else
        UNDEFINED;
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBR0_EL2, Translation Table Base Register 0 (EL2)

The TTBR0_EL2 characteristics are:

Purpose

When [HCR_EL2.E2H](#) is 0, holds the base address of the translation table for the initial lookup for stage 1 of an address translation in the EL2 translation regime, and other information for this translation regime.

When [HCR_EL2.E2H](#) is 1, holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the lower VA range in the EL2&0 translation regime, and other information for this translation regime.

Configuration

AArch64 System register TTBR0_EL2 bits [47:1] are architecturally mapped to AArch32 System register [HTTBR\[47:1\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

TTBR0_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | BADDR[47:1] | | | | | | | | | | | | | | | |
| BADDR[47:1] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | CnP |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

When [FEAT_VHE](#) is implemented:

When [HCR_EL2.E2H](#) is 0, this field is RES0.

When [HCR_EL2.E2H](#) is 1, it holds an ASID for the translation table base address. The [TCR_EL2.A1](#) field selects either TTBR0_EL2.ASID or TTBR1_EL2.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BADDR[47:1], bits [47:1]

Translation table base address:

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The smallest permitted value of x is 6. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of [TCR_EL2.T0SZ](#), the translation stage, and the translation granule size.

Note

A translation table is required to be aligned to the size of the table. If a table contains fewer than eight entries, it must be aligned on a 64 byte address boundary.

If the value of [TCR_EL2.{I}PS](#) is not 0b110, then:

- Register bits[($x-1$):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits A[51:48] of the stage 1 translation table base address are 0b0000.

If FEAT_LPA is implemented and the value of [TCR_EL2.{I}PS](#) is 0b110, then:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
 - Register bit[1] is RES0.
 - When $x > 6$, register bits[($x-1$):6] are RES0.
-

Note

The OA size specified by [TCR_EL2.{I}PS](#) is determined as follows:

- The value of [TCR_EL2.PS](#) when the value of [HCR_EL2.E2H](#) is 0.
- The value of [TCR_EL2.IPS](#) when the value of [HCR_EL2.E2H](#) is 1.

[TCR_EL2.{I}PS](#) == 0b110 is permitted when:

- FEAT_LPA is implemented and the 64KB translation granule is used.
- FEAT_LPA2 is implemented and the 4KB or 16KB translation granule is used.

When the value of [ID_AA64MMFR0_EL1.PARange](#) indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register when the Effective value of [TCR_EL2.{I}PS](#) is 0b110 and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR0_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[($x-1$):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL2.CnP is 1.

| CnP | Meaning |
|-----|--|
| 0b0 | <p>The translation table entries pointed to by TTBR0_EL2 for the current translation regime, and ASID if applicable, are permitted to differ from corresponding entries for TTBR0_EL2 for other PEs in the Inner Shareable domain. This is not affected by:</p> <ul style="list-style-type: none"> The value of TTBR0_EL2.CnP on those other PEs. When the current translation regime is the EL2&0 regime, the value of the current ASID. |
| 0b1 | <p>The translation table entries pointed to by TTBR0_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL2.CnP is 1 and all of the following apply:</p> <ul style="list-style-type: none"> The translation table entries are pointed to by TTBR0_EL2. The translation tables relate to the same translation regime. If that translation regime is the EL2&0 regime, the ASID is the same as the current ASID. |

This bit is permitted to be cached in a TLB.

Note

If the value of the TTBR0_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL2s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are **CONSTRAINED UNPREDICTABLE**, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR0_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic TTBR0_EL2 or TTBR0_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TTBR0_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return TTBR0_EL2;
elsif PSTATE.EL == EL3 then
    return TTBR0_EL2;

```

MSR TTBR0_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TTBR0_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    TTBR0_EL2 = X[t];

```

MRS <Xt>, TTBR0_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.TTBR0_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x200];
    else
        return TTBR0_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TTBR0_EL2;
    else
        return TTBR0_EL1;
elsif PSTATE.EL == EL3 then
    return TTBR0_EL1;

```

MSR TTBR0_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.TTBR0_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x200] = X[t];
    else
        TTBR0_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        TTBR0_EL2 = X[t];
    else
        TTBR0_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TTBR0_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBR0_EL3, Translation Table Base Register 0 (EL3)

The TTBR0_EL3 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of an address translation in the EL3 translation regime, and other information for this translation regime.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to TTBR0_EL3 are UNDEFINED.

Attributes

TTBR0_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | BADDR[47:1] | | | | | | | | | | | | | | | |
| BADDR[47:1] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | CnP |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

BADDR[47:1], bits [47:1]

Translation table base address:

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The smallest permitted value of x is 6. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of [TCR_EL3.T0SZ](#), the translation stage, and the translation granule size.

Note

A translation table is required to be aligned to the size of the table. If a table contains fewer than eight entries, it must be aligned on a 64 byte address boundary.

If the value of [TCR_EL3.PS](#) is not 0b110, then:

- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits A[51:48] of the stage 1 translation table base address are 0b0000.

If FEAT_LPA is implemented and the value of [TCR_EL3.PS](#) is 0b110, then:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
- Register bit[1] is RES0.
- When x>6, register bits[(x-1):6] are RES0.

Note

[TCR_EL3](#).PS==0b110 is permitted when:

- FEAT_LPA is implemented and the 64KB translation granule is used.
- FEAT_LPA2 is implemented and the 4KB or 16KB translation granule is used.

When the value of [ID_AA64MMFR0_EL1](#).PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register when the Effective value of [TCR_EL3](#).PS is 0b110 and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR0_EL3, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[(x-1):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TTBR0_EL3 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1.

| CnP | Meaning |
|-----|---|
| 0b0 | The translation table entries pointed to by TTBR0_EL3, for the current translation regime, are permitted to differ from corresponding entries for TTBR0_EL3 for other PEs in the Inner Shareable domain. This is not affected by the value of TTBR0_EL3.CnP on those other PEs. |
| 0b1 | The translation table entries pointed to by TTBR0_EL3 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0_EL3.CnP is 1 and the translation table entries are pointed to by TTBR0_EL3. |

This bit is permitted to be cached in a TLB.

Note

If the value of the TTBR0_EL3.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0_EL3s do not point to the same translation table entries the results of translations using TTBR0_EL3 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR0_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TTBR0_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0010 | 0b0000 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return TTBR0_EL3;
```

MSR TTBR0_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0010 | 0b0000 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    TTBR0_EL3 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBR1_EL1, Translation Table Base Register 1 (EL1)

The TTBR1_EL1 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the higher VA range in the EL1&0 stage 1 translation regime, and other information for this translation regime.

Configuration

AArch64 System register TTBR1_EL1 bits [63:0] are architecturally mapped to AArch32 System register [TTBR1\[63:0\]](#).

Attributes

TTBR1_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | BADDR[47:1] | | | | | | | | | | | | | | | |
| BADDR[47:1] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | CnP |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

An ASID for the translation table base address. The [TCR_EL1.A1](#) field selects either TTBR0_EL1.ASID or TTBR1_EL1.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR[47:1], bits [47:1]

Translation table base address:

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The smallest permitted value of x is 6. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of [TCR_EL1.T1SZ](#), the translation stage, and the translation granule size.

Note

A translation table is required to be aligned to the size of the table. If a table contains fewer than eight entries, it must be aligned on a 64 byte address boundary.

If the value of [TCR_EL1.IPS](#) is not 0b110, then:

- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits A[51:48] of the stage 1 translation table base address are 0b0000.

If FEAT_LPA is implemented and the value of [TCR_EL1](#).IPS is 0b110, then:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
- Register bit[1] is RES0.
- When $x > 6$, register bits[($x-1$):6] are RES0.

Note

[TCR_EL1](#).IPS==0b110 is permitted when:

- FEAT_LPA is implemented and the 64KB translation granule is used.
- FEAT_LPA2 is implemented and the 4KB or 16KB translation granule is used.

When the value of [ID_AA64MMFR0_EL1](#).PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register when the Effective value of [TCR_EL1](#).IPS is 0b110 and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR1_EL1, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[($x-1$):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TBR1_EL1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1.

| CnP | Meaning |
|-----|---|
| 0b0 | <p>The translation table entries pointed to by TTBR1_EL1, for the current translation regime and ASID, are permitted to differ from corresponding entries for TTBR1_EL1 for other PEs in the Inner Shareable domain. This is not affected by:</p> <ul style="list-style-type: none"> • The value of TTBR1_EL1.CnP on those other PEs. • The value of the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID. |
| 0b1 | <p>The translation table entries pointed to by TTBR1_EL1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1_EL1.CnP is 1 and all of the following apply:</p> <ul style="list-style-type: none"> • The translation table entries are pointed to by TTBR1_EL1. • The translation tables relate to the same translation regime. • The ASID is the same as the current ASID. • If EL2 is implemented and enabled in the current Security state, the value of the current VMID. |

This bit is permitted to be cached in a TLB.

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR1_EL1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1_EL1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR1_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic TTBR1_EL1 or TTBR1_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TTBR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.TTBR1_EL1 == '1'
    then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x210];
    else
        return TTBR1_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TTBR1_EL2;
    else
        return TTBR1_EL1;
elsif PSTATE.EL == EL3 then
    return TTBR1_EL1;

```

MSR TTBR1_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.TTBR1_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x210] = X[t];
    else
        TTBR1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        TTBR1_EL2 = X[t];
    else
        TTBR1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TTBR1_EL1 = X[t];

```

MRS <Xt>, TTBR1_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0010 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x210];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TTBR1_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return TTBR1_EL1;
    else
        UNDEFINED;

```

MSR TTBR1_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0010 | 0b0000 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x210] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        TTBR1_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        TTBR1_EL1 = X[t];
    else
        UNDEFINED;
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBR1_EL2, Translation Table Base Register 1 (EL2)

The TTBR1_EL2 characteristics are:

Purpose

When [HCR_EL2.E2H](#) is 1, holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the higher VA range in the EL2&0 translation regime, and other information for this translation regime.

Note

When [HCR_EL2.E2H](#) is 0, the contents of this register are ignored by the PE, except for a direct read or write of the register.

Configuration

This register is present only when FEAT_VHE is implemented. Otherwise, direct accesses to TTBR1_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

TTBR1_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ASID | | | | | | | | | | | | | | | | BADDR[47:1] | | | | | | | | | | | | | | | |
| BADDR[47:1] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | CnP |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ASID, bits [63:48]

An ASID for the translation table base address. The [TCR_EL2.A1](#) field selects either TTBR0_EL2.ASID or TTBR1_EL2.ASID.

If the implementation has only 8 bits of ASID, then the upper 8 bits of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR[47:1], bits [47:1]

Translation table base address:

- Bits A[47:x] of the stage 1 translation table base address bits are in register bits[47:x].
- Bits A[(x-1):0] of the stage 1 translation table base address are zero.

Address bit x is the minimum address bit required to align the translation table to the size of the table. The smallest permitted value of x is 6. The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of [TCR_EL2.T1SZ](#), the translation stage, and the translation granule size.

Note

A translation table is required to be aligned to the size of the table. If a table contains fewer than eight entries, it must be aligned on a 64 byte address boundary.

If the value of [TCR_EL2](#).{I}PS is not 0b110, then:

- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs, then bits A[51:48] of the stage 1 translation table base address are 0b0000.

If FEAT_LPA is implemented and the value of [TCR_EL2](#).{I}PS is 0b110, then:

- Bits A[51:48] of the stage 1 translation table base address bits are in register bits[5:2].
 - Register bit[1] is RES0.
 - When x>6, register bits[(x-1):6] are RES0.
-

Note

The OA size specified by [TCR_EL2](#).{I}PS is determined as follows:

- The value of [TCR_EL2](#).PS when the value of [HCR_EL2](#).E2H is 0.
- The value of [TCR_EL2](#).IPS when the value of [HCR_EL2](#).E2H is 1.

[TCR_EL2](#).{I}PS==0b110 is permitted when:

- FEAT_LPA is implemented and the 64KB translation granule is used.
- FEAT_LPA2 is implemented and the 4KB or 16KB translation granule is used.

When the value of [ID_AA64MMFR0_EL1](#).PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register when the Effective value of [TCR_EL2](#).{I}PS is 0b110 and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If any register bit[47:1] that is defined as RES0 has the value 1 when a translation table walk is done using TTBR1_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits A[(x-1):0] of the stage 1 translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by TBR1_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1_EL2.CnP is 1.

| CnP | Meaning |
|-----|--|
| 0b0 | The translation table entries pointed to by TTBR1_EL2 for the current ASID are permitted to differ from corresponding entries for TTBR1_EL2 for other PEs in the Inner Shareable domain. This is not affected by: <ul style="list-style-type: none"> The value of TTBR1_EL2.CnP on those other PEs. The value of the current ASID. |
| 0b1 | The translation table entries pointed to by TTBR1_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1_EL2.CnP is 1 and all of the following apply: <ul style="list-style-type: none"> The translation table entries are pointed to by TTBR1_EL2. The ASID is the same as the current ASID. |

This bit is permitted to be cached in a TLB.

Note

- TTBR1_EL2 is accessible only when the value of [HCR_EL2.E2H](#) is 1, meaning the current translation regime is the EL2&0 regime.
- If the value of the TTBR1_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1_EL2s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR1_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic TTBR1_EL2 or TTBR1_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, TTBR1_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return TTBR1_EL2;
elsif PSTATE.EL == EL3 then
    return TTBR1_EL2;

```

MSR TTBR1_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|-----|-----|-----|-----|-----|
|-----|-----|-----|-----|-----|

| | | | | |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0000 | 0b001 |
|------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    TTBR1_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    TTBR1_EL2 = X[t];

```

MRS <Xt>, TTBR1_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TRVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.TTBR1_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x210];
    else
        return TTBR1_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return TTBR1_EL2;
    else
        return TTBR1_EL1;
elsif PSTATE.EL == EL3 then
    return TTBR1_EL1;

```

MSR TTBR1_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.TVM == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.TTBR1_EL1 == '1'
then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x210] = X[t];
    else
        TTBR1_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        TTBR1_EL2 = X[t];
    else
        TTBR1_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    TTBR1_EL1 = X[t];

```


UAO, User Access Override

The UAO characteristics are:

Purpose

Allows access to the User Access Override bit.

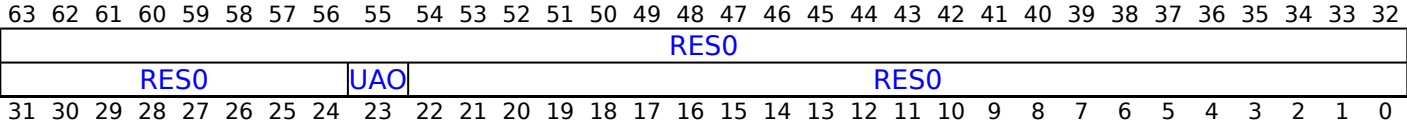
Configuration

This register is present only when FEAT_UAO is implemented. Otherwise, direct accesses to UAO are UNDEFINED.

Attributes

UAO is a 64-bit register.

Field descriptions



Bits [63:24]

Reserved, RES0.

UAO, bit [23]

User Access Override.

| UAO | Meaning |
|-----|---|
| 0b0 | The behavior of LDTR* and STTR* instructions is as defined in the base Armv8 architecture. |
| 0b1 | When executed at EL1, or at EL2 with HCR_EL2 .{E2H, TGE} == {1, 1}, LDTR* and STTR* instructions behave as the equivalent LDR* and STR* instructions. |

When executed at EL3, or at EL2 with [HCR_EL2](#).E2H == 0 or [HCR_EL2](#).TGE == 0, the LDTR* and STTR* instructions behave as the equivalent LDR* and STR* instructions, regardless of the setting of the PSTATE.UAO bit.

Bits [22:0]

Reserved, RES0.

Accessing UAO

For more information about the operation of the MSR (immediate) accessor, see 'MSR (immediate)'.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, UAO

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0010 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    return Zeros(40):PSTATE.UA0:Zeros(23);
elsif PSTATE.EL == EL2 then
    return Zeros(40):PSTATE.UA0:Zeros(23);
elsif PSTATE.EL == EL3 then
    return Zeros(40):PSTATE.UA0:Zeros(23);

```

MSR UA0, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0100 | 0b0010 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    PSTATE.UA0 = X[t]<23>;
elsif PSTATE.EL == EL2 then
    PSTATE.UA0 = X[t]<23>;
elsif PSTATE.EL == EL3 then
    PSTATE.UA0 = X[t]<23>;

```

MSR UA0, #<imm>

| op0 | op1 | CRn | op2 |
|------|-------|--------|-------|
| 0b00 | 0b000 | 0b0100 | 0b011 |

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VBAR_EL1, Vector Base Address Register (EL1)

The VBAR_EL1 characteristics are:

Purpose

Holds the vector base address for any exception that is taken to EL1.

Configuration

AArch64 System register VBAR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [VBAR\[31:0\]](#).

Attributes

VBAR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Vector Base Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Vector Base Address | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:11]

Vector Base Address. Base address of the exception vectors for exceptions taken to EL1.

Note

If the implementation does not support FEAT_LVA, then:

- If tagged addresses are being used, bits [55:48] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:48] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.

If the implementation supports FEAT_LVA, then:

- If tagged addresses are being used, bits [55:52] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:52] of VBAR_EL1 must be the same or else the use of the vector address will result in a recursive exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [10:0]

Reserved, RES0.

Accessing VBAR_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic VBAR_EL1 or VBAR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, VBAR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x250];
    else
        return VBAR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return VBAR_EL2;
    else
        return VBAR_EL1;
elsif PSTATE.EL == EL3 then
    return VBAR_EL1;

```

MSR VBAR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x250] = X[t];
    else
        VBAR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        VBAR_EL2 = X[t];
    else
        VBAR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    VBAR_EL1 = X[t];

```

MRS <Xt>, VBAR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x250];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return VBAR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        return VBAR_EL1;
    else
        UNDEFINED;

```

MSR VBAR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x250] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        VBAR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        VBAR_EL1 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VBAR_EL2, Vector Base Address Register (EL2)

The VBAR_EL2 characteristics are:

Purpose

Holds the vector base address for any exception that is taken to EL2.

Configuration

AArch64 System register VBAR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [HVBAR\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VBAR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Vector Base Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Vector Base Address | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:11]

Vector Base Address. Base address of the exception vectors for exceptions taken to EL2.

Note

If FEAT_LVA is implemented:

- If [HCR_EL2.E2H](#) == 0b1:
 - If tagged addresses are being used, bits [55:52] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
 - If tagged addresses are not being used, bits [63:52] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
- If [HCR_EL2.E2H](#) == 0b0:
 - If tagged addresses are being used, bits [55:52] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.
 - If tagged addresses are not being used, bits [63:52] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.

If FEAT_LVA is not implemented:

- If [HCR_EL2.E2H](#) == 0b1:
 - If tagged addresses are being used, bits [55:48] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
 - If tagged addresses are not being used, bits [63:48] of VBAR_EL2 must be the same or else the use of the vector address will result in a recursive exception.
- If [HCR_EL2.E2H](#) == 0b0:

-
- If tagged addresses are being used, bits [55:48] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.
 - If tagged addresses are not being used, bits [63:48] of VBAR_EL2 must be 0 or else the use of the vector address will result in a recursive exception.
-

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [10:0]

Reserved, RES0.

Accessing VBAR_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic VBAR_EL2 or VBAR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, VBAR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VBAR_EL2;
elsif PSTATE.EL == EL3 then
    return VBAR_EL2;

```

MSR VBAR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VBAR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VBAR_EL2 = X[t];

```

MRS <Xt>, VBAR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x250];
    else
        return VBAR_EL1;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        return VBAR_EL2;
    else
        return VBAR_EL1;
elsif PSTATE.EL == EL3 then
    return VBAR_EL1;

```

MSR VBAR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '011' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGWTR_EL2.VBAR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x250] = X[t];
    else
        VBAR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        VBAR_EL2 = X[t];
    else
        VBAR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    VBAR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VBAR_EL3, Vector Base Address Register (EL3)

The VBAR_EL3 characteristics are:

Purpose

Holds the vector base address for any exception that is taken to EL3.

Configuration

This register is present only when EL3 is implemented. Otherwise, direct accesses to VBAR_EL3 are UNDEFINED.

Attributes

VBAR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Vector Base Address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Vector Base Address | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:11]

Vector Base Address. Base address of the exception vectors for exceptions taken to EL3.

Note

If the implementation does not support FEAT_LVA, then:

- If tagged addresses are being used, bits [55:48] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:48] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.

If the implementation supports FEAT_LVA, then:

- If tagged addresses are being used, bits [55:52] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.
- If tagged addresses are not being used, bits [63:52] of VBAR_EL3 must be 0 or else the use of the vector address will result in a recursive exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [10:0]

Reserved, RES0.

Accessing VBAR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, VBAR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return VBAR_EL3;

```

MSR VBAR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    VBAR_EL3 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VDISR_EL2, Virtual Deferred Interrupt Status Register

The VDISR_EL2 characteristics are:

Purpose

Records that a virtual SError interrupt has been consumed by an ESB instruction executed at EL1.

An indirect write to VDISR_EL2 made by an ESB instruction does not require an explicit synchronization operation for the value written to be observed by a direct read of [DISR_EL1](#) or [DISR](#) occurring in program order after the ESB instruction.

Configuration

AArch64 System register VDISR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [VDISR\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to VDISR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VDISR_EL2 is a 64-bit register.

Field descriptions

When EL1 is using AArch64:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|----|----|----|----|----|----|----|----|----|-----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| A | RES0 | | | | | | | | | | IDS | ISS | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:25]

Reserved, RES0.

IDS, bit [24]

The value copied from [VSESR_EL2.IDS](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]

The value copied from [VSESR_EL2](#).ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When EL1 is using AArch32 and VDISR_EL2.LPAE == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|------|-----|------|-------|------|------|----|----|----|----|----|---------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| A | RES0 | | | | | | | | | | | | | | | AET | RES0 | ExT | RES0 | FS[4] | LPAE | RES0 | | | | | | FS[3:0] | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

The value copied from [VSESR_EL2](#).AET.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

The value copied from [VSESR_EL2](#).ExT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

FS, bits [10, 3:0]

Fault status code. Set to 0b10110 when an ESB instruction defers a virtual SError interrupt.

| FS | Meaning |
|---------|--------------------------------|
| 0b10110 | Asynchronous SError interrupt. |

All other values are reserved.

The FS field is split as follows:

- FS[4] is VDISR_EL2[10].
- FS[3:0] is VDISR_EL2[3:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

Format.

Set to [TTBCR](#).EAE when an ESB instruction defers a virtual SError interrupt.

| LPAE | Meaning |
|------|--|
| 0b0 | Using the Short-descriptor translation table format. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:4]

Reserved, RES0.

When EL1 is using AArch32 and VDISR_EL2.LPAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|------|-----|------|------|------|--------|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| A | RES0 | | | | | | | | | | | | | | AET | RES0 | EXT | RES0 | LPAE | RES0 | STATUS | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

The value copied from [VSESR_EL2](#).AET.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

The value copied from [VSESR_EL2](#).ExT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

Format.

Set to [TTBCR](#).EAE when an ESB instruction defers a virtual SError interrupt.

| LPAE | Meaning |
|------|---|
| 0b1 | Using the Long-descriptor translation table format. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status code. Set to 0b010001 when an ESB instruction defers a virtual SError interrupt.

| STATUS | Meaning |
|----------|--------------------------------|
| 0b010001 | Asynchronous SError interrupt. |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VDISR_EL2

An indirect write to VDISR_EL2 made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of [DISR_EL1](#) or [DISR](#) occurring in program order after the ESB instruction.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, VDISR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x500];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VDISR_EL2;
elsif PSTATE.EL == EL3 then
    return VDISR_EL2;

```

MSR VDISR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b1100 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x500] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VDISR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VDISR_EL2 = X[t];

```

MRS <Xt>, DISR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AMO == '1' then
        return VDISR_EL2;
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return Zeros();
    else
        return DISR_EL1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return Zeros();
    else
        return DISR_EL1;
elsif PSTATE.EL == EL3 then
    return DISR_EL1;

```

MSR DISR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b1100 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.AMO == '1' then
        VDISR_EL2 = X[t];
    elsif HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        //no operation
    else
        DISR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !Halted() && SCR_EL3.EA == '1' then
        //no operation
    else
        DISR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    DISR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VMPIDR_EL2, Virtualization Multiprocessor ID Register

The VMPIDR_EL2 characteristics are:

Purpose

Holds the value of the Virtualization Multiprocessor ID. This is the value returned by EL1 reads of [MPIDR_EL1](#).

Configuration

AArch64 System register VMPIDR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [VMPIDR\[31:0\]](#).

If EL2 is not implemented, reads of this register return the value of the [MPIDR_EL1](#) and writes to the register are ignored.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VMPIDR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|------|----|----|----|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|------|------|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | Aff3 | | | | | | | |
| RES1 | U | RES0 | | | | | | MT | Aff2 | | | | | | | | Aff1 | | | | | | | | Aff0 | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3. See the description of VMPIDR_EL2.Aff0 for more information.

Aff3 is not supported in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [31]

Reserved, RES1.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

| U | Meaning |
|-----|---|
| 0b0 | Processor is part of a multiprocessor system. |
| 0b1 | Processor is part of a uniprocessor system. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using a multithreading type approach. See the description of VMPIDR_EL2.Aff0 for more information about affinity levels.

| MT | Meaning |
|-----|---|
| 0b0 | Performance of PEs at the lowest affinity level is largely independent. |
| 0b1 | Performance of PEs at the lowest affinity level is very interdependent. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Aff2, bits [23:16]

Affinity level 2. See the description of VMPIDR_EL2.Aff0 for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Aff1, bits [15:8]

Affinity level 1. See the description of VMPIDR_EL2.Aff0 for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Aff0, bits [7:0]

Affinity level 0. This is the affinity level that is most significant for determining PE behavior. Higher affinity levels are increasingly less significant in determining PE behavior. The assigned value of the MPIDR.{Aff2, Aff1, Aff0} or [MPIDR_EL1](#).{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VMPIDR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, VMPIDR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0000 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x050];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VMPIDR_EL2;
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        return MPIDR_EL1;
    else
        return VMPIDR_EL2;

```

MSR VMPIDR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0000 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x050] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VMPIDR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        //no operation
    else
        VMPIDR_EL2 = X[t];

```

MRS <Xt>, MPIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.MPIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() then
        return VMPIDR_EL2;
    else
        return MPIDR_EL1;
elsif PSTATE.EL == EL2 then
    return MPIDR_EL1;
elsif PSTATE.EL == EL3 then
    return MPIDR_EL1;

```


VNCR_EL2, Virtual Nested Control Register

The VNCR_EL2 characteristics are:

Purpose

When FEAT_NV2 is implemented, holds the base address that is used to define the memory location that is accessed by transformed reads and writes of System registers.

Configuration

This register is present only when FEAT_NV2 is implemented. Otherwise, direct accesses to VNCR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VNCR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RESS | | | | | | | | | | | BADDR | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | BADDR | | | | | | | | | | | RES0 | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

RESS, bits [63:53]

Reserved, Sign extended. If the bits marked as RESS do not all have the same value, then there is a CONSTRAINED UNPREDICTABLE choice between:

- Generating an EL2 translation regime Translation abort on use of the VNCR_EL2 register.
- Bits[63:49] of VNCR_EL2 are treated as the same value as bit[48] for all purposes other than reading back the register.
- Bits[63:49] of VNCR_EL2 are treated as the same value as bit[48] for all purposes.
- If the virtual address space for EL2 supports more than 48 bits, bits[63:53] of VNCR_EL2 are treated as the same value as bit[52] for all purposes other than reading back the register.
- If the virtual address space for EL2 supports more than 48 bits, bits[63:53] of VNCR_EL2 are treated as the same value as bit[52].

Where the EL2 translation regime has upper and lower address ranges, bit[52] is used to select between those address ranges to determine if the address space supports more than 48 bits.

BADDR, bits [52:12]

Base Address. If the virtual address space for EL2 does not support more than 48 bits, then bits [52:49] are RESS.

When a register read/write is transformed to be a Load or Store, the address of the load/store is to SignOffset(VNCR_EL2.BADDR:Offset<11:0>, 64).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

Accessing VNCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, VNCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x0B0];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VNCR_EL2;
elsif PSTATE.EL == EL3 then
    return VNCR_EL2;

```

MSR VNCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x0B0] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VNCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VNCR_EL2 = X[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VPIDR_EL2, Virtualization Processor ID Register

The VPIDR_EL2 characteristics are:

Purpose

Holds the value of the Virtualization Processor ID. This is the value returned by EL1 reads of [MIDR_EL1](#).

Configuration

AArch64 System register VPIDR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [VPIDR\[31:0\]](#).

If EL2 is not implemented, reads of this register return the value of the [MIDR_EL1](#) and writes to the register are ignored.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VPIDR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|---------|----|----|----|--------------|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Implementer | | | | | | | | Variant | | | | Architecture | | | | PartNum | | | | | | | | | | | | Revision | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

| Implementer | Meaning |
|-------------|--|
| 0x00 | Reserved for software use. |
| 0x41 | Arm Limited. |
| 0x42 | Broadcom Corporation. |
| 0x43 | Cavium Inc. |
| 0x44 | Digital Equipment Corporation. |
| 0x46 | Fujitsu Ltd. |
| 0x49 | Infineon Technologies AG. |
| 0x4D | Motorola or Freescale Semiconductor Inc. |
| 0x4E | NVIDIA Corporation. |
| 0x50 | Applied Micro Circuits Corporation. |
| 0x51 | Qualcomm Inc. |
| 0x56 | Marvell International Ltd. |
| 0x69 | Intel Corporation. |
| 0xC0 | Ampere Computing. |

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Variant, bits [23:20]

An IMPLEMENTATION DEFINED variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Architecture, bits [19:16]

Architecture version. Defined values are:

| Architecture | Meaning |
|--------------|---|
| 0b0001 | Armv4. |
| 0b0010 | Armv4T. |
| 0b0011 | Armv5 (obsolete). |
| 0b0100 | Armv5T. |
| 0b0101 | Armv5TE. |
| 0b0110 | Armv5TEJ. |
| 0b0111 | Armv6. |
| 0b1111 | Architectural features are individually identified in the ID_* registers. |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PartNum, bits [15:4]

An IMPLEMENTATION DEFINED primary part number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Revision, bits [3:0]

An IMPLEMENTATION DEFINED revision number for the device.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VPIDR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, VPIDR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0000 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x088];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VPIDR_EL2;
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        return MIDR_EL1;
    else
        return VPIDR_EL2;

```

MSR VPIDR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0000 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x088] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VPIDR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        //no operation
    else
        VPIDR_EL2 = X[t];

```

MRS <Xt>, MIDR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0000 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    if IsFeatureImplemented(FEAT_IDST) then
        if EL2Enabled() && HCR_EL2.TGE == '1' then
            AArch64.SystemAccessTrap(EL2, 0x18);
        else
            AArch64.SystemAccessTrap(EL1, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && (!HaveEL(EL3) || SCR_EL3.FGTEn == '1') && HFGTR_EL2.MIDR_EL1 == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    elsif EL2Enabled() then
        return VPIDR_EL2;
    else
        return MIDR_EL1;
elsif PSTATE.EL == EL2 then
    return MIDR_EL1;
elsif PSTATE.EL == EL3 then
    return MIDR_EL1;

```


VSESR_EL2, Virtual SError Exception Syndrome Register

The VSESR_EL2 characteristics are:

Purpose

Provides the syndrome value reported to software on taking a virtual SError interrupt exception to EL1, or on executing an ESB instruction at EL1.

When the virtual SError interrupt injected using [HCR_EL2.VSE](#) is taken to EL1 using AArch64, then the syndrome value is reported in [ESR_EL1](#).

When the virtual SError interrupt injected using [HCR_EL2.VSE](#) is taken to EL1 using AArch32, then the syndrome value is reported in [DFSR](#), {AET, ExT} and the remainder of [DFSR](#) is set as defined by VMSAv8-32. For more information, see The AArch32 Virtual Memory System Architecture.

When the virtual SError interrupt injected using [HCR_EL2.VSE](#) is deferred by an ESB instruction, then the syndrome value is written to [VDISR_EL2](#).

Configuration

AArch64 System register VSESR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [VDFSR\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to VSESR_EL2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VSESR_EL2 is a 64-bit register.

Field descriptions

When EL1 is using AArch32:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|------|-----|------|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | AET | RES0 | EXT | RES0 | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:16]

Reserved, RES0.

AET, bits [15:14]

When a virtual SError interrupt is taken to EL1 using AArch32, [DFSR](#)[15:4] is set to VSESR_EL2.AET.

When a virtual SError interrupt is deferred by an ESB instruction, [VDISR_EL2](#)[15:4] is set to VSESR_EL2.AET.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

When a virtual SError interrupt is taken to EL1 using AArch32, [DFSR](#)[12] is set to VSESR_EL2.ExT.

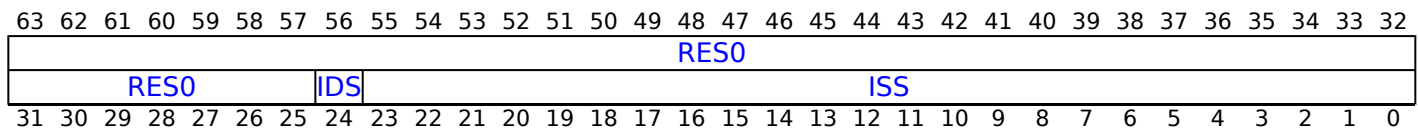
When a virtual SError interrupt is deferred by an ESB instruction, [VDISR_EL2](#)[12] is set to VSESR_EL2.ExT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

When EL1 is using AArch64:**Bits [63:25]**

Reserved, RES0.

IDS, bit [24]

When a virtual SError interrupt is taken to EL1 using AArch64, [ESR_EL1](#)[24] is set to VSESR_EL2.IDS.

When a virtual SError interrupt is deferred by an ESB instruction, [VDISR_EL2](#)[24] is set to VSESR_EL2.IDS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [23:0]

When a virtual SError interrupt is taken to EL1 using AArch64, [ESR_EL1](#)[23:0] is set to VSESR_EL2.ISS.

When a virtual SError interrupt is deferred by an ESB instruction, [VDISR_EL2](#)[23:0] is set to VSESR_EL2.ISS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VSESR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, VSESR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x508];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VSESR_EL2;
elsif PSTATE.EL == EL3 then
    return VSESR_EL2;

```

MSR VSESR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0101 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x508] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VSESR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VSESR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VSTCR_EL2, Virtualization Secure Translation Control Register

The VSTCR_EL2 characteristics are:

Purpose

The control register for stage 2 of the Secure EL1&0 translation regime.

Configuration

This register is present only when FEAT_SEL2 is implemented. Otherwise, direct accesses to VSTCR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VSTCR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|-----|------|----|----|----|----|----|-----|------|----|----|----|----|----|----|----|--|-----|------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | SL2 | RES0 |
| RES1 | SA | SW | RES0 | | | | | | | | | | | | | TG0 | RES0 | | | | | | SLO | TOSZ | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | |

Any of the bits in VSTCR_EL2 are permitted to be cached in a TLB.

Bits [63:34]

Reserved, RES0.

SL2, bit [33]

When FEAT_LPA2 is implemented:

Starting level of the Secure stage 2 translation lookup controlled by VSTCR_EL2.

If [VTCR_EL2.DS](#) == 1, then VSTCR_EL2.SL2, in combination with VSTCR_EL2.SL0, gives encodings for the Secure stage 2 translation table walk initial lookup level.

If [VTCR_EL2.DS](#) == 0, then VSTCR_EL2.SL2 is RES0.

If the translation granule size is not 4KB, then this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [32]

Reserved, RES0.

Bit [31]

Reserved, RES1.

SA, bit [30]

Secure stage 2 translation output address space.

| SA | Meaning |
|-----|---|
| 0b0 | All stage 2 translations for the Secure IPA space access the Secure PA space. |
| 0b1 | All stage 2 translations for the Secure IPA space access the Non-secure PA space. |

When the value of VSTCR_EL2.SW is 1, this bit behaves as 1 for all purposes other than reading back the value of the bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SW, bit [29]

Secure stage 2 translation address space.

| SW | Meaning |
|-----|--|
| 0b0 | All stage 2 translation table walks for the Secure IPA space are to the Secure PA space. |
| 0b1 | All stage 2 translation table walks for the Secure IPA space are to the Non-secure PA space. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [28:16]

Reserved, RES0.

TG0, bits [15:14]

Secure stage 2 granule size for [VSTTBR_EL2](#).

| TG0 | Meaning |
|------|---------|
| 0b00 | 4KB. |
| 0b01 | 64KB. |
| 0b10 | 16KB. |

Other values are reserved.

If FEAT_GTG is implemented, [ID_AA64MMFR0_EL1](#).{TGran4_2, TGran16_2, TGran64_2} indicate which granule sizes are supported for stage 2 translation.

If FEAT_GTG is not implemented, [ID_AA64MMFR0_EL1](#).{TGran4, TGran16, TGran64} indicate which granule sizes are supported.

If the value is programmed to either a reserved value, or a size that has not been implemented, then for all purposes other than read back from this register, the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [13:8]

Reserved, RES0.

SL0, bits [7:6]**When FEAT_TTST is implemented:**

Starting level of the Secure stage 2 translation lookup, controlled by VSTCR_EL2. The meaning of this field depends on the value of VSTCR_EL2.TG0.

| SL0 | Meaning |
|------|--|
| 0b00 | If VSTCR_EL2.TG0 is 0b00 (4KB granule): <ul style="list-style-type: none"> • If FEAT_LPA2 is not implemented, start at level 2. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is 0b0, start at level 2. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is 0b1, start at level -1. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3. |
| 0b01 | If VSTCR_EL2.TG0 is 0b00 (4KB granule): <ul style="list-style-type: none"> • If FEAT_LPA2 is not implemented, start at level 1. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is 0b0, start at level 1. • If FEAT_LPA2 is implemented, the combination of VSTCR_EL2.SL0 == 01 and VSTCR_EL2.SL2 == 1 is reserved. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2. |
| 0b10 | If VSTCR_EL2.TG0 is 0b00 (4KB granule): <ul style="list-style-type: none"> • If FEAT_LPA2 is not implemented, start at level 0. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is 0b0, start at level 0. • If FEAT_LPA2 is implemented, the combination of VSTCR_EL2.SL0 == 10 and VSTCR_EL2.SL2 == 1 is reserved. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1. |
| 0b11 | If VSTCR_EL2.TG0 is 0b00 (4KB granule): <ul style="list-style-type: none"> • If FEAT_LPA2 is not implemented, start at level 3. • If FEAT_LPA2 is implemented and VSTCR_EL2.SL2 is 0b0, start at level 3. • If FEAT_LPA2 is implemented, the combination of VSTCR_EL2.SL0 == 11 and VSTCR_EL2.SL2 == 1 is reserved. If VSTCR_EL2.TG0 is 0b10 (16KB granule) and FEAT_LPA2 is implemented, start at level 0. |

If this field is programmed to a value that is not consistent with the programming of VSTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Starting level of the Secure stage 2 translation lookup, controlled by VSTCR_EL2. The meaning of this field depends on the value of VSTCR_EL2.TG0.

| SL0 | Meaning |
|------|---|
| 0b00 | If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 2. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3. |
| 0b01 | If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 1. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2. |
| 0b10 | If VSTCR_EL2.TG0 is 0b00 (4KB granule), start at level 0. If VSTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1. |

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VSTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T0SZ, bits [5:0]

The size offset of the memory region addressed by [VSTTBR_EL2](#). The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for this field depend on the level of translation table and the memory translation granule size, as described in the AArch64 Virtual Memory System Architecture chapter.

If this field is programmed to a value that is not consistent with the programming of SL0, then a stage 2 level 0 Translation fault is generated.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VSTCR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, VSTCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elseif EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x048];
    elseif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        return VSTCR_EL2;
elseif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        return VSTCR_EL2;

```

MSR VSTCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elseif EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x048] = X[t];
    elseif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        VSTCR_EL2 = X[t];
elseif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        VSTCR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VSTTBR_EL2, Virtualization Secure Translation Table Base Register

The VSTTBR_EL2 characteristics are:

Purpose

The base register for stage 2 of the Secure EL1&0 translation regime. Holds the base address of the translation table for the initial lookup for stage 2 of an address translation in the Secure EL1&0 translation regime, and other information for this translation stage.

Configuration

This register is present only when FEAT_SEL2 is implemented. Otherwise, direct accesses to VSTTBR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VSTTBR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | BADDR | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | BADDR | | | | | | | | | | | | | | | CnP |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

BADDR, bits [47:1]

Translation table base address, A[47:x] or A[51:x].

Note

A translation table must be aligned to the size of the table, except that when using a translation table base address larger than 48 bits the minimum alignment of a table containing fewer than eight entries is 64 bytes.

If the value of [VTCR_EL2](#).PS is 0b110, then:

- Register bits[47:z] hold bits[47:z] of the stage 1 translation table base address, where z is determined as follows:
 - If $x \geq 6$ then $z=x$.
 - Otherwise, $z=6$.
- Register bits[5:2] hold bits[51:48] of the stage 1 translation table base address.
- When $z > x$ register bits[(z-1):x] are RES0, and bits[(z-1):x] of the translation table base address are zero.
- When $x > 6$ register bits[(x-1):6] are RES0.
- Register bit[1] is RES0.
- Bits[5:2] of the stage 1 translation table base address are zero.

Note

When the value of [ID_AA64MMFR0_EL1.PARange](#) indicates that the implementation does not support a 52-bit PA size, if a translation table lookup uses this register with the 64KB translation granule when the Effective value of [VTCR_EL2.PS](#) is 0b110 and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If the Effective value of [VTCR_EL2.PS](#) is not 0b110, then:

- Register bits[47:x] hold bits[47:x] of the stage 1 translation table base address.
- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs then bits[51:48] of the translation table base addresses used in this stage of translation are 0b0000.

If any VSTTBR_EL2[47:1] bit that is defined as RES0 has the value 1 when a translation table walk is performed using VSTTBR_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of [VSTCR_EL2.T0SZ](#), the stage of translation, and the translation granule size.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

Common not Private, for stage 2 of the Secure EL1&0 translation regime. In an implementation that includes FEAT_TTCNP, indicates whether each entry that is pointed to by VSTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1.

| CnP | Meaning |
|-----|--|
| 0b0 | The translation table entries pointed to by VSTTBR_EL2 are permitted to differ from the entries for VSTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID. |
| 0b1 | The translation table entries pointed to by VSTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VSTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID. |

This bit is permitted to be cached in a TLB.

Note

If the value of VSTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VSTTBR_EL2s do not point to the same translation table entries when using the current VMID, then the results of translations using VSTTBR_EL2 are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

When this register has an architecturally-defined reset value, this field resets to a value that is architecturally UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VSTTBR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, VSTTBR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x030];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        return VSTTBR_EL2;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        return VSTTBR_EL2;

```

MSR VSTTBR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x030] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if !IsSecure() then
        UNDEFINED;
    else
        VSTTBR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.EEL2 == '0' then
        UNDEFINED;
    else
        VSTTBR_EL2 = X[t];

```


VTCR_EL2, Virtualization Translation Control Register

The VTCR_EL2 characteristics are:

Purpose

The control register for stage 2 of the EL1&0 translation regime.

Configuration

AArch64 System register VTCR_EL2 bits [31:0] are architecturally mapped to AArch32 System register [VTCR\[31:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VTCR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|-----|-------|-------|-------|-------|------|----|----|------|----|----|-----|-----|-------|-------|-----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES1 | NSA | NSW | HWU62 | HWU61 | HWU60 | HWU59 | RES0 | HD | HA | RES0 | VS | PS | TG0 | SH0 | ORGNO | IRGN0 | SL0 | T0SZ | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Any of the bits in VTCR_EL2 are permitted to be cached in a TLB.

Bits [63:34]

Reserved, RES0.

SL2, bit [33]

When FEAT_LPA2 is implemented:

Starting level of the stage 2 translation lookup controlled by VTCR_EL2.

If VTCR_EL2.DS == 1, then VTCR_EL2.SL2, in combination with VTCR_EL2.SL0, gives encodings for the stage 2 translation table walk initial lookup level.

If VTCR_EL2.DS == 0, then VTCR_EL2.SL2 is RES0.

If the translation granule size is not 4KB, then this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DS, bit [32]**When FEAT_LPA2 is implemented:**

This field affects 52-bit output addressing when using 4KB and 16KB translation granules in stage 2 of the EL1&0 translation regime.

| DS | Meaning |
|---|---|
| 0b0 | <p>Bits[49:48] of translation descriptors are RES0.</p> <p>Bits[9:8] in block and page descriptors encode shareability information in the SH[1:0] field. Bits[9:8] in table descriptors are ignored by hardware.</p> <p>The minimum value of VTCR_EL2.T0SZ is 16. Any memory access using a smaller value generates a stage 2 level 0 translation table fault.</p> <p>The minimum value of VSTCR_EL2.T0SZ is 16. Any memory access using a smaller value generates a stage 2 level 0 translation table fault.</p> <p>Output address[51:48] is 0000.</p> |
| 0b1 | <p>Bits[49:48] of translation descriptors hold output address[49:48].</p> <p>Bits[9:8] in translation descriptors hold output address[51:50].</p> <p>The shareability information of block and page descriptors for cacheable locations is determined by VTCR_EL2.SH0.</p> <p>The minimum value of VTCR_EL2.T0SZ is 12. Any memory access using a smaller value generates a stage 2 level 0 translation table fault.</p> <p>The minimum value of VSTCR_EL2.T0SZ is 12. Any memory access using a smaller value generates a stage 2 level 0 translation table fault.</p> |
| <p>Note</p> <p>As FEAT_LPA must be implemented if VTCR_EL2.DS == 1, the minimum values of VTCR_EL2.T0SZ and VSTCR_EL2.T0SZ are 12, as determined by that extension.</p> <p>For the TLBI range instructions affecting IPA, the format of the argument is changed so that bits[36:0] hold BaseADDR[52:16]. For the 4KB translation granule, bits[15:12] of BaseADDR are treated as 0000. For the 16KB translation granule, bits[15:14] of BaseADDR are treated as 00.</p> <p>Note</p> <p>This forces alignment of the ranges used by the TLBI range instructions.</p> | |

This field is RES0 for a 64KB translation granule.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [31]

Reserved, RES1.

NSA, bit [30]**When FEAT_SEL2 is implemented:**

Non-secure stage 2 translation output address space for the Secure EL1&0 translation regime.

| NSA | Meaning |
|-----|--|
| 0b0 | All stage 2 translations for the Non-secure IPA space of the Secure EL1&0 translation regime access the Secure PA space. |
| 0b1 | All stage 2 translations for the Non-secure IPA space of the Secure EL1&0 translation regime access the Non-secure PA space. |

This bit behaves as 1 for all purposes other than reading back the value of the bit when one of the following is true:

- The value of VTCR_EL2.NSW is 1.
- The value of [VSTCR_EL2.SA](#) is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSW, bit [29]

When FEAT_SEL2 is implemented:

Non-secure stage 2 translation table address space for the Secure EL1&0 translation regime.

| NSW | Meaning |
|-----|---|
| 0b0 | All stage 2 translation table walks for the Non-secure IPA space of the Secure EL1&0 translation regime are to the Secure PA space. |
| 0b1 | All stage 2 translation table walks for the Non-secure IPA space of the Secure EL1&0 translation regime are to the Non-secure PA space. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HWU62, bit [28]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 2 translation table Block or Page entry.

| HWU62 | Meaning |
|-------|---|
| 0b0 | Bit[62] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[62] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU61, bit [27]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 2 translation table Block or Page entry.

| HWU61 | Meaning |
|-------|---|
| 0b0 | Bit[61] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[61] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU60, bit [26]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 2 translation table Block or Page entry.

| HWU60 | Meaning |
|-------|---|
| 0b0 | Bit[60] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[60] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU59, bit [25]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 2 translation table Block or Page entry.

| HWU59 | Meaning |
|-------|---|
| 0b0 | Bit[59] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[59] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bits [24:23]

Reserved, RES0.

HD, bit [22]

When FEAT_HAFDBS is implemented:

Hardware management of dirty state in stage 2 translations when EL2 is enabled in the current Security state.

| HD | Meaning |
|-----|---|
| 0b0 | Stage 2 hardware management of dirty state disabled. |
| 0b1 | Stage 2 hardware management of dirty state enabled, only if the VTCR_EL2.HA bit is also set to 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HA, bit [21]

When FEAT_HAFDBS is implemented:

Hardware Access flag update in stage 2 translations when EL2 is enabled in the current Security state.

| HA | Meaning |
|-----|--------------------------------------|
| 0b0 | Stage 2 Access flag update disabled. |
| 0b1 | Stage 2 Access flag update enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [20]

Reserved, RES0.

VS, bit [19]

When FEAT_VMID16 is implemented:

VMID Size.

| VS | Meaning |
|-----|---|
| 0b0 | 8-bit VMID. The upper 8 bits of VTTBR_EL2 are ignored by the hardware, and treated as if they are all zeros, for every purpose except when reading back the register. |
| 0b1 | 16-bit VMID. The upper 8 bits of VTTBR_EL2 are used for allocation and matching in the TLB. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PS, bits [18:16]

Physical address Size for the second stage of translation.

| PS | Meaning |
|-------|-----------------|
| 0b000 | 32 bits, 4GB. |
| 0b001 | 36 bits, 64GB. |
| 0b010 | 40 bits, 1TB. |
| 0b011 | 42 bits, 4TB. |
| 0b100 | 44 bits, 16TB. |
| 0b101 | 48 bits, 256TB. |
| 0b110 | 52 bits, 4PB. |

All other values are reserved.

The reserved values behave in the same way as the 0b101 or 0b110 encoding, but software must not rely on this property as the behavior of the reserved values might change in a future revision of the architecture.

If the translation granule is not 64KB and FEAT_LPA2 is not implemented, the value 0b110 is treated as reserved.

It is IMPLEMENTATION DEFINED whether an implementation that does not implement FEAT_LPA supports setting the value of 0b110 for the 64KB translation granule size or whether setting this value behaves as the 0b101 encoding.

In an implementation that supports 52-bit PAs, if the value of this field is not 0b110 or a value treated as 0b110, then bits[51:48] of every translation table base address for the stage of translation controlled by VTCR_EL2 are 0b0000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TG0, bits [15:14]

Granule size for the [VTTBR_EL2](#).

| TG0 | Meaning |
|------|---------|
| 0b00 | 4KB. |
| 0b01 | 64KB. |
| 0b10 | 16KB. |

Other values are reserved.

If FEAT_GTG is implemented, [ID_AA64MMFR0_EL1](#).{TGran4_2, TGran16_2, TGran64_2} indicate which granule sizes are supported for stage 2 translation.

If FEAT_GTG is not implemented, [ID_AA64MMFR0_EL1](#).{TGran4, TGran16, TGran64} indicate which granule sizes are supported.

If the value is programmed to either a reserved value or a size that has not been implemented, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the sizes that has been implemented for all purposes other than the value read back from this register.

It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the value that corresponds to the size chosen.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using [VTTBR_EL2](#) or [VSTTBR_EL2](#).

| SH0 | Meaning |
|------|------------------|
| 0b00 | Non-shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGNO, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using [VTTBR_EL2](#) or [VSTTBR_EL2](#).

| ORGNO | Meaning |
|-------|---|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGNO, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using [VTTBR_EL2](#) or [VSTTBR_EL2](#).

| IRGNO | Meaning |
|-------|---|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SL0, bits [7:6]

When FEAT_TTST is implemented:

Starting level of the stage 2 translation lookup, controlled by VTCR_EL2. The meaning of this field depends on the value of VTCR_EL2.TG0.

| SL0 | Meaning |
|------|---|
| 0b00 | If VTCR_EL2.TG0 is 0b00 (4KB granule): <ul style="list-style-type: none"> • If FEAT_LPA2 is not implemented, start at level 2. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is 0b0, start at level 2. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is 0b1, start at level -1. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3. |
| 0b01 | If VTCR_EL2.TG0 is 0b00 (4KB granule): <ul style="list-style-type: none"> • If FEAT_LPA2 is not implemented, start at level 1. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is 0b0, start at level 1. • If FEAT_LPA2 is implemented, the combination of VTCR_EL2.SL0 == 01 and VTCR_EL2.SL2 == 1 is reserved. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2. |
| 0b10 | If VTCR_EL2.TG0 is 0b00 (4KB granule): <ul style="list-style-type: none"> • If FEAT_LPA2 is not implemented, start at level 0. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is 0b0, start at level 0. • If FEAT_LPA2 is implemented, the combination of VTCR_EL2.SL0 == 10 and VTCR_EL2.SL2 == 1 is reserved. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1. |
| 0b11 | If VTCR_EL2.TG0 is 0b00 (4KB granule): <ul style="list-style-type: none"> • If FEAT_LPA2 is not implemented, start at level 3. • If FEAT_LPA2 is implemented and VTCR_EL2.SL2 is 0b0, start at level 3. • If FEAT_LPA2 is implemented, the combination of VTCR_EL2.SL0 == 11 and VTCR_EL2.SL2 == 1 is reserved. If VTCR_EL2.TG0 is 0b10 (16KB granule) and FEAT_LPA2 is implemented, start at level 0. |

If this field is programmed to a value that is not consistent with the programming of VTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Starting level of the stage 2 translation lookup, controlled by VTCR_EL2. The meaning of this field depends on the value of VTCR_EL2.TG0.

| SL0 | Meaning |
|------|---|
| 0b00 | If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 2. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 3. |
| 0b01 | If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 1. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 2. |
| 0b10 | If VTCR_EL2.TG0 is 0b00 (4KB granule), start at level 0. If VTCR_EL2.TG0 is 0b10 (16KB granule) or 0b01 (64KB granule), start at level 1. |

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of VTCR_EL2.T0SZ, then a stage 2 level 0 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T0SZ, bits [5:0]

The size offset of the memory region addressed by [VTTBR_EL2](#). The region size is $2^{(64-T0SZ)}$ bytes.

The maximum and minimum possible values for T0SZ depend on the level of translation table and the memory translation granule size, as described in 'The AArch64 Virtual Memory System Architecture'.

If this field is programmed to a value that is not consistent with the programming of SL0, then a stage 2 level 0 Translation fault is generated.

Note

For the 4KB translation granule, if FEAT_LPA2 is implemented and this field is less than 16, the translation table walk begins with a level -1 initial lookup.

For the 16KB translation granule, if FEAT_LPA2 is implemented and this field is less than 17, the translation table walk begins with a level 0 initial lookup.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VTCR_EL2

Any of the bits in VTCR_EL2 are permitted to be cached in a TLB.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, VTCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x040];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VTCR_EL2;
elsif PSTATE.EL == EL3 then
    return VTCR_EL2;

```

MSR VTCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0001 | 0b010 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x040] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VTCR_EL2 = X[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VTTBR_EL2, Virtualization Translation Table Base Register

The VTTBR_EL2 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 2 of an address translation in the EL1&0 translation regime, and other information for this translation regime.

Configuration

AArch64 System register VTTBR_EL2 bits [63:0] are architecturally mapped to AArch32 System register [VTTBR\[63:0\]](#).

If EL2 is not implemented, this register is RES0 from EL3.

This register has no effect if EL2 is not enabled in the current Security state.

Attributes

VTTBR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| VMID | | | | | | | | | | | | | | | | BADDR | | | | | | | | | | | | | | | | |
| BADDR | | | | | | | | | | | | | | | | CnP | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

VMID, bits [63:48]

VMID encoding when FEAT_VMID16 is implemented and VTCR_EL2.VS == 1

| | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VMID | | | | | | | | | | | | | | | |

VMID, bits [15:0]

The VMID for the translation table.

If the implementation has an 8-bit VMID, bits [15:8] of this field are RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VMID encoding when FEAT_VMID16 is not implemented or VTCR_EL2.VS == 0

| | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|---|---|------|---|---|---|---|---|---|---|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | VMID | | | | | | | |

Bits [15:8]

Reserved, RES0.

VMID, bits [7:0]

The VMID for the translation table.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- [VTCR_EL2](#).VS is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR, bits [47:1]

Translation table base address, A[47:x] or A[51:x], bits[47:1].

Note

A translation table must be aligned to the size of the table, except that when using a translation table base address larger than 48 bits the minimum alignment of a table containing fewer than eight entries is 64 bytes.

In an implementation that includes FEAT_LPA, if the value of [VTCR_EL2](#).PS is 0b110, then:

- Register bits[47:z] hold bits[47:z] of the stage 1 translation table base address, where z is determined as follows:
 - If $x \geq 6$ then $z=x$.
 - Otherwise, $z=6$.
- Register bits[5:2] hold bits[51:48] of the stage 1 translation table base address.
- When $z > x$ register bits[(z-1):x] are RES0, and bits[(z-1):x] of the translation table base address are zero.
- When $x > 6$ register bits[(x-1):6] are RES0.
- Register bit[1] is RES0.
- Bits[5:2] of the stage 1 translation table base address are zero.
- In an implementation that includes FEAT_TTCNP, bit[0] of the stage 1 translation table base address is zero.

Note

When the value of [ID_AA64MMFR0_EL1](#).PARange indicates that the implementation does not support a 52 bit PA size, if a translation table lookup uses this register when the Effective value of [VTCR_EL2](#).PS is 0b110 and the value of register bits[5:2] is nonzero, an Address size fault is generated.

If the Effective value of [VTCR_EL2](#).PS is not 0b110 then:

- Register bits[47:x] hold bits[47:x] of the stage 1 translation table base address.
- Register bits[(x-1):1] are RES0.
- If the implementation supports 52-bit PAs and IPAs then bits[51:48] of the translation table base addresses used in this stage of translation are 0b0000.

If any VTTBR_EL2[47:0] bit that is defined as RES0 has the value 1 when a translation table walk is performed using VTTBR_EL2, then the translation table base address might be misaligned, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Bits[x-1:0] of the translation table base address are treated as if all the bits are zero. The value read back from the corresponding register bits is either the value written to the register or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The AArch64 Virtual Memory System Architecture chapter describes how x is calculated based on the value of [VTCR_EL2.T0SZ](#), the stage of translation, and the translation granule size.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by VTTBR_EL2 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VTTBR_EL2.CnP is 1.

| CnP | Meaning |
|-----|--|
| 0b0 | The translation table entries pointed to by VTTBR_EL2 are permitted to differ from the entries for VTTBR_EL2 for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID. |
| 0b1 | The translation table entries pointed to by VTTBR_EL2 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VTTBR_EL2.CnP is 1 and the VMID is the same as the current VMID. |

This bit is permitted to be cached in a TLB.

Note

If the value of VTTBR_EL2.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VTTBR_EL2s do not point to the same translation table entries when using the current VMID then the results of translations using VTTBR_EL2 are CONstrained UNPREDICTABLE, see 'CONstrained UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing VTTBR_EL2

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, VTTBR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        return NVMem[0x020];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VTTBR_EL2;
elsif PSTATE.EL == EL3 then
    return VTTBR_EL2;

```

MSR VTTBR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0010 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV> == '11' then
        NVMem[0x020] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTTBR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    VTTBR_EL2 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ZCR_EL1, SVE Control Register (EL1)

The ZCR_EL1 characteristics are:

Purpose

This register controls aspects of SVE visible at Exception levels EL1 and EL0.

Configuration

This register is present only when FEAT_SVE is implemented. Otherwise, direct accesses to ZCR_EL1 are UNDEFINED.

This register has no effect if the PE is in Streaming SVE mode.

When [HCR_EL2](#).{E2H, TGE} == {1, 1} and EL2 is enabled in the current Security state, this register has no effect on execution at EL0.

Attributes

ZCR_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | RAZ/WI | | | | | | | | LEN | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Effective SVE Vector Length (VL).

Constrains the effective scalable vector register length for EL1 and EL0 to (LEN+1)*128 bits. VL only takes effect when the PE is not in Streaming SVE mode.

An implementation is permitted to include any set of vector lengths that are multiples of 128 bits, from 128 bits to 2048 bits inclusive, and required to support all vector lengths that are powers of two, from 128 bits up to its maximum implemented vector length.

For all purposes other than returning the result of a direct read of ZCR_EL1, this field selects the effective vector length as follows:

- If the requested length is larger than the effective vector length at the next more privileged Exception level in the current Security state, if any, then the effective vector length at the more privileged Exception level is used.
- If the requested length is not implemented, then the requested length rounded down to the nearest implemented scalable vector length is used.
- Otherwise, the requested length is used.

An indirect read of ZCR_EL1.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ZCR_EL1

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL3 using the mnemonic ZCR_EL1 or ZCR_EL12 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ZCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif CPACR_EL1.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x1E0];
    else
        return ZCR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif HCR_EL2.E2H == '1' then
        return ZCR_EL2;
    else
        return ZCR_EL1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        return ZCR_EL1;

```

MSR ZCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif CPACR_EL1.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x1E0] = X[t];
    else
        ZCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif HCR_EL2.E2H == '1' then
        ZCR_EL2 = X[t];
    else
        ZCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL1 = X[t];

```

MRS <Xt>, ZCR_EL12

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0001 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        return NVMem[0x1E0];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.EZ == '0' then
            UNDEFINED;
        elsif CPTR_EL2.ZEN == 'x0' then
            AArch64.SystemAccessTrap(EL2, 0x19);
        elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x19);
        else
            return ZCR_EL1;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        if CPTR_EL3.EZ == '0' then
            AArch64.SystemAccessTrap(EL3, 0x19);
        else
            return ZCR_EL1;
    else
        UNDEFINED;

```

MSR ZCR_EL12, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b101 | 0b0001 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '101' then
        NVMem[0x1E0] = X[t];
    elsif EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if HCR_EL2.E2H == '1' then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.EZ == '0' then
            UNDEFINED;
        elsif CPTR_EL2.ZEN == 'x0' then
            AArch64.SystemAccessTrap(EL2, 0x19);
        elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.SystemAccessTrap(EL3, 0x19);
        else
            ZCR_EL1 = X[t];
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' then
        if CPTR_EL3.EZ == '0' then
            AArch64.SystemAccessTrap(EL3, 0x19);
        else
            ZCR_EL1 = X[t];
    else
        UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ZCR_EL2, SVE Control Register (EL2)

The ZCR_EL2 characteristics are:

Purpose

This register controls aspects of SVE visible at Exception levels EL2, EL1, and EL0.

Configuration

This register is present only when FEAT_SVE is implemented. Otherwise, direct accesses to ZCR_EL2 are UNDEFINED.

This register has no effect if EL2 is not enabled in the current Security state, or if the PE is in Streaming SVE mode.

Attributes

ZCR_EL2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|-----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | RAZ/WI | | | | | | LEN | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Effective SVE Vector Length (VL).

Constrains the effective scalable vector register length for EL2, EL1, and EL0 to (LEN+1)*128 bits when EL2 is enabled in the current Security state. VL only takes effect when the PE is not in Streaming SVE mode.

An implementation is permitted to include any set of vector lengths that are multiples of 128 bits, from 128 bits to 2048 bits inclusive, and required to support all vector lengths that are powers of two, from 128 bits up to its maximum implemented vector length.

For all purposes other than returning the result of a direct read of ZCR_EL2, this field selects the effective vector length as follows:

- If the requested length is larger than the effective vector length at the next more privileged Exception level in the current Security state, if any, then the effective vector length at the more privileged Exception level is used.
- If the requested length is not implemented, then the requested length rounded down to the nearest implemented scalable vector length is used.
- Otherwise, the requested length is used.

An indirect read of ZCR_EL2.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ZCR_EL2

When [HCR_EL2.E2H](#) is 1, without explicit synchronization, access from EL2 using the mnemonic ZCR_EL2 or ZCR_EL1 are not guaranteed to be ordered with respect to accesses using the other mnemonic.

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ZCR_EL2

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    else
        return ZCR_EL2;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        return ZCR_EL2;

```

MSR ZCR_EL2, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x18);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL2 = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL2 = X[t];

```

MRS <Xt>, ZCR_EL1

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif CPACR_EL1.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        return NVMem[0x1E0];
    else
        return ZCR_EL1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif HCR_EL2.E2H == '1' then
        return ZCR_EL2;
    else
        return ZCR_EL1;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        return ZCR_EL1;

```

MSR ZCR_EL1, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif CPACR_EL1.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL1, 0x19);
    elsif EL2Enabled() && HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif EL2Enabled() && HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif EL2Enabled() && HCR_EL2.<NV2,NV1,NV> == '111' then
        NVMem[0x1E0] = X[t];
    else
        ZCR_EL1 = X[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && CPTR_EL3.EZ == '0' then
        UNDEFINED;
    elsif HCR_EL2.E2H == '0' && CPTR_EL2.TZ == '1' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HCR_EL2.E2H == '1' && CPTR_EL2.ZEN == 'x0' then
        AArch64.SystemAccessTrap(EL2, 0x19);
    elsif HaveEL(EL3) && CPTR_EL3.EZ == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.SystemAccessTrap(EL3, 0x19);
    elsif HCR_EL2.E2H == '1' then
        ZCR_EL2 = X[t];
    else
        ZCR_EL1 = X[t];
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL1 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ZCR_EL3, SVE Control Register (EL3)

The ZCR_EL3 characteristics are:

Purpose

This register controls aspects of SVE visible at all Exception levels.

Configuration

This register is present only when FEAT_SVE is implemented. Otherwise, direct accesses to ZCR_EL3 are UNDEFINED.

This register has no effect if the PE is in Streaming SVE mode.

Attributes

ZCR_EL3 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|--|--|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | RAZ/WI | | | | | | | | LEN | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | |

Bits [63:9]

Reserved, RES0.

Bits [8:4]

Reserved, RAZ/WI.

LEN, bits [3:0]

Effective SVE Vector Length (VL).

Constrains the effective scalable vector register length for all Exception levels to (LEN+1)*128 bits. VL only takes effect when the PE is not in Streaming SVE mode.

An implementation is permitted to include any set of vector lengths that are multiples of 128 bits, from 128 bits to 2048 bits inclusive, and required to support all vector lengths that are powers of two, from 128 bits up to its maximum implemented vector length.

For all purposes other than returning the result of a direct read of ZCR_EL3, this field selects the effective vector length as follows:

- If the requested length is not implemented, then the requested length rounded down to the nearest implemented scalable vector length is used.
- Otherwise, the requested length is used.

An indirect read of ZCR_EL3.LEN appears to occur in program order relative to a direct write of the same register, without the need for explicit synchronization.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ZCR_EL3

Accesses to this register use the following encodings in the System register encoding space:

MRS <Xt>, ZCR_EL3

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        return ZCR_EL3;

```

MSR ZCR_EL3, <Xt>

| op0 | op1 | CRn | CRm | op2 |
|------|-------|--------|--------|-------|
| 0b11 | 0b110 | 0b0001 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CPTR_EL3.EZ == '0' then
        AArch64.SystemAccessTrap(EL3, 0x19);
    else
        ZCR_EL3 = X[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AArch32 System Registers

[ACTLR](#): Auxiliary Control Register

[ACTLR2](#): Auxiliary Control Register 2

[ADFSR](#): Auxiliary Data Fault Status Register

[AIDR](#): Auxiliary ID Register

[AIFSR](#): Auxiliary Instruction Fault Status Register

[AMAIRO](#): Auxiliary Memory Attribute Indirection Register 0

[AMAIR1](#): Auxiliary Memory Attribute Indirection Register 1

[AMCFGR](#): Activity Monitors Configuration Register

[AMCGCR](#): Activity Monitors Counter Group Configuration Register

[AMCNTENCLR0](#): Activity Monitors Count Enable Clear Register 0

[AMCNTENCLR1](#): Activity Monitors Count Enable Clear Register 1

[AMCNTENSET0](#): Activity Monitors Count Enable Set Register 0

[AMCNTENSET1](#): Activity Monitors Count Enable Set Register 1

[AMCR](#): Activity Monitors Control Register

[AMEVCNTR0<n>](#): Activity Monitors Event Counter Registers 0

[AMEVCNTR1<n>](#): Activity Monitors Event Counter Registers 1

[AMEVTYPER0<n>](#): Activity Monitors Event Type Registers 0

[AMEVTYPER1<n>](#): Activity Monitors Event Type Registers 1

[AMUSERENR](#): Activity Monitors User Enable Register

[APSR](#): Application Program Status Register

[CCSIDR](#): Current Cache Size ID Register

[CCSIDR2](#): Current Cache Size ID Register 2

[CLIDR](#): Cache Level ID Register

[CNTFRQ](#): Counter-timer Frequency register

[CNTHCTL](#): Counter-timer Hyp Control register

[CNTHPS_CTL](#): Counter-timer Secure Physical Timer Control Register (EL2)

[CNTHPS_CVAL](#): Counter-timer Secure Physical Timer CompareValue Register (EL2)

[CNTHPS_TVAL](#): Counter-timer Secure Physical Timer TimerValue Register (EL2)

[CNTHP_CTL](#): Counter-timer Hyp Physical Timer Control register

[CNTHP_CVAL](#): Counter-timer Hyp Physical CompareValue register

[CNTHP_TVAL](#): Counter-timer Hyp Physical Timer TimerValue register

[CNTHVS_CTL](#): Counter-timer Secure Virtual Timer Control Register (EL2)

[CNTHVS_CVAL](#): Counter-timer Secure Virtual Timer CompareValue Register (EL2)

[CNTHVS_TVAL](#): Counter-timer Secure Virtual Timer TimerValue Register (EL2)

[CNTHV_CTL](#): Counter-timer Virtual Timer Control register (EL2)
[CNTHV_CVAL](#): Counter-timer Virtual Timer CompareValue register (EL2)
[CNTHV_TVAL](#): Counter-timer Virtual Timer TimerValue register (EL2)
[CNTKCTL](#): Counter-timer Kernel Control register
[CNTPCT](#): Counter-timer Physical Count register
[CNTPCTSS](#): Counter-timer Self-Synchronized Physical Count register
[CNTP_CTL](#): Counter-timer Physical Timer Control register
[CNTP_CVAL](#): Counter-timer Physical Timer CompareValue register
[CNTP_TVAL](#): Counter-timer Physical Timer TimerValue register
[CNTVCT](#): Counter-timer Virtual Count register
[CNTVCTSS](#): Counter-timer Self-Synchronized Virtual Count register
[CNTVOFF](#): Counter-timer Virtual Offset register
[CNTV_CTL](#): Counter-timer Virtual Timer Control register
[CNTV_CVAL](#): Counter-timer Virtual Timer CompareValue register
[CNTV_TVAL](#): Counter-timer Virtual Timer TimerValue register
[CONTEXTIDR](#): Context ID Register
[CPACR](#): Architectural Feature Access Control Register
[CPSR](#): Current Program Status Register
[CSSELR](#): Cache Size Selection Register
[CTR](#): Cache Type Register
[DACR](#): Domain Access Control Register
[DBGAUTHSTATUS](#): Debug Authentication Status register
[DBGBCR<n>](#): Debug Breakpoint Control Registers
[DBGBVR<n>](#): Debug Breakpoint Value Registers
[DBGBXVR<n>](#): Debug Breakpoint Extended Value Registers
[DBGCLAIMCLR](#): Debug CLAIM Tag Clear register
[DBGCLAIMSET](#): Debug CLAIM Tag Set register
[DBGDCCINT](#): DCC Interrupt Enable Register
[DBGDEVID](#): Debug Device ID register 0
[DBGDEVID1](#): Debug Device ID register 1
[DBGDEVID2](#): Debug Device ID register 2
[DBGDIDR](#): Debug ID Register
[DBGDRAR](#): Debug ROM Address Register
[DBGDSAR](#): Debug Self Address Register
[DBGDSCRext](#): Debug Status and Control Register, External View
[DBGDSCRint](#): Debug Status and Control Register, Internal View

[DBGDTRRText](#): Debug OS Lock Data Transfer Register, Receive, External View

[DBGDTRRXint](#): Debug Data Transfer Register, Receive

[DBGDTRTXext](#): Debug OS Lock Data Transfer Register, Transmit

[DBGDTRTXint](#): Debug Data Transfer Register, Transmit

[DBGOSDLR](#): Debug OS Double Lock Register

[DBGOSECCR](#): Debug OS Lock Exception Catch Control Register

[DBGOSLAR](#): Debug OS Lock Access Register

[DBGOSLSR](#): Debug OS Lock Status Register

[DBGPRCR](#): Debug Power Control Register

[DBGVCR](#): Debug Vector Catch Register

[DBGWCR<n>](#): Debug Watchpoint Control Registers

[DBGWFAR](#): Debug Watchpoint Fault Address Register

[DBGWVR<n>](#): Debug Watchpoint Value Registers

[DFAR](#): Data Fault Address Register

[DFSR](#): Data Fault Status Register

[DISR](#): Deferred Interrupt Status Register

[DLR](#): Debug Link Register

[DSPSR](#): Debug Saved Program Status Register

[ELR_hyp](#): Exception Link Register (Hyp mode)

[ERRIDR](#): Error Record ID Register

[ERRSELR](#): Error Record Select Register

[ERXADDR](#): Selected Error Record Address Register

[ERXADDR2](#): Selected Error Record Address Register 2

[ERXCTLR](#): Selected Error Record Control Register

[ERXCTLR2](#): Selected Error Record Control Register 2

[ERXFR](#): Selected Error Record Feature Register

[ERXFR2](#): Selected Error Record Feature Register 2

[ERXMISC0](#): Selected Error Record Miscellaneous Register 0

[ERXMISC1](#): Selected Error Record Miscellaneous Register 1

[ERXMISC2](#): Selected Error Record Miscellaneous Register 2

[ERXMISC3](#): Selected Error Record Miscellaneous Register 3

[ERXMISC4](#): Selected Error Record Miscellaneous Register 4

[ERXMISC5](#): Selected Error Record Miscellaneous Register 5

[ERXMISC6](#): Selected Error Record Miscellaneous Register 6

[ERXMISC7](#): Selected Error Record Miscellaneous Register 7

[ERXSTATUS](#): Selected Error Record Primary Status Register

[FCSEIDR](#): FCSE Process ID register

[FPEXC](#): Floating-Point Exception Control register

[FPSCR](#): Floating-Point Status and Control Register

[FPSID](#): Floating-Point System ID register

[HACR](#): Hyp Auxiliary Configuration Register

[HACTLR](#): Hyp Auxiliary Control Register

[HACTLR2](#): Hyp Auxiliary Control Register 2

[HADESR](#): Hyp Auxiliary Data Fault Status Register

[HAIFSR](#): Hyp Auxiliary Instruction Fault Status Register

[HAMAIRO](#): Hyp Auxiliary Memory Attribute Indirection Register 0

[HAMAIR1](#): Hyp Auxiliary Memory Attribute Indirection Register 1

[HCPTR](#): Hyp Architectural Feature Trap Register

[HCR](#): Hyp Configuration Register

[HCR2](#): Hyp Configuration Register 2

[HDCR](#): Hyp Debug Control Register

[HDFAR](#): Hyp Data Fault Address Register

[HIFAR](#): Hyp Instruction Fault Address Register

[HMAIRO](#): Hyp Memory Attribute Indirection Register 0

[HMAIR1](#): Hyp Memory Attribute Indirection Register 1

[HPFAR](#): Hyp IPA Fault Address Register

[HRMR](#): Hyp Reset Management Register

[HSCTLR](#): Hyp System Control Register

[HSR](#): Hyp Syndrome Register

[HSTR](#): Hyp System Trap Register

[HTCR](#): Hyp Translation Control Register

[HTPIDR](#): Hyp Software Thread ID Register

[HTRFCR](#): Hyp Trace Filter Control Register

[HTTBR](#): Hyp Translation Table Base Register

[HVBAR](#): Hyp Vector Base Address Register

[ICC_AP0R<n>](#): Interrupt Controller Active Priorities Group 0 Registers

[ICC_AP1R<n>](#): Interrupt Controller Active Priorities Group 1 Registers

[ICC_ASGI1R](#): Interrupt Controller Alias Software Generated Interrupt Group 1 Register

[ICC_BPR0](#): Interrupt Controller Binary Point Register 0

[ICC_BPR1](#): Interrupt Controller Binary Point Register 1

[ICC_CTLR](#): Interrupt Controller Control Register

[ICC_DIR](#): Interrupt Controller Deactivate Interrupt Register

[ICC_EOIRO](#): Interrupt Controller End Of Interrupt Register 0

[ICC_EOIR1](#): Interrupt Controller End Of Interrupt Register 1

[ICC_HPPIRO](#): Interrupt Controller Highest Priority Pending Interrupt Register 0

[ICC_HPPIR1](#): Interrupt Controller Highest Priority Pending Interrupt Register 1

[ICC_HSRE](#): Interrupt Controller Hyp System Register Enable register

[ICC_IAR0](#): Interrupt Controller Interrupt Acknowledge Register 0

[ICC_IAR1](#): Interrupt Controller Interrupt Acknowledge Register 1

[ICC_IGRPEN0](#): Interrupt Controller Interrupt Group 0 Enable register

[ICC_IGRPEN1](#): Interrupt Controller Interrupt Group 1 Enable register

[ICC_MCTLR](#): Interrupt Controller Monitor Control Register

[ICC_MGRPEN1](#): Interrupt Controller Monitor Interrupt Group 1 Enable register

[ICC_MSRE](#): Interrupt Controller Monitor System Register Enable register

[ICC_PMR](#): Interrupt Controller Interrupt Priority Mask Register

[ICC_RPR](#): Interrupt Controller Running Priority Register

[ICC_SGI0R](#): Interrupt Controller Software Generated Interrupt Group 0 Register

[ICC_SGI1R](#): Interrupt Controller Software Generated Interrupt Group 1 Register

[ICC_SRE](#): Interrupt Controller System Register Enable register

[ICH_AP0R<n>](#): Interrupt Controller Hyp Active Priorities Group 0 Registers

[ICH_AP1R<n>](#): Interrupt Controller Hyp Active Priorities Group 1 Registers

[ICH_EISR](#): Interrupt Controller End of Interrupt Status Register

[ICH_ELRSR](#): Interrupt Controller Empty List Register Status Register

[ICH_HCR](#): Interrupt Controller Hyp Control Register

[ICH_LR<n>](#): Interrupt Controller List Registers

[ICH_LRC<n>](#): Interrupt Controller List Registers

[ICH_MISR](#): Interrupt Controller Maintenance Interrupt State Register

[ICH_VMCR](#): Interrupt Controller Virtual Machine Control Register

[ICH_VTR](#): Interrupt Controller VGIC Type Register

[ICV_AP0R<n>](#): Interrupt Controller Virtual Active Priorities Group 0 Registers

[ICV_AP1R<n>](#): Interrupt Controller Virtual Active Priorities Group 1 Registers

[ICV_BPR0](#): Interrupt Controller Virtual Binary Point Register 0

[ICV_BPR1](#): Interrupt Controller Virtual Binary Point Register 1

[ICV_CTLR](#): Interrupt Controller Virtual Control Register

[ICV_DIR](#): Interrupt Controller Deactivate Virtual Interrupt Register

[ICV_EOIRO](#): Interrupt Controller Virtual End Of Interrupt Register 0

[ICV_EOIR1](#): Interrupt Controller Virtual End Of Interrupt Register 1

[ICV_HPPIRO](#): Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0

[ICV_HPPIR1](#): Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1

[ICV_IAR0](#): Interrupt Controller Virtual Interrupt Acknowledge Register 0

[ICV_IAR1](#): Interrupt Controller Virtual Interrupt Acknowledge Register 1

[ICV_IGRPEN0](#): Interrupt Controller Virtual Interrupt Group 0 Enable register

[ICV_IGRPEN1](#): Interrupt Controller Virtual Interrupt Group 1 Enable register

[ICV_PMR](#): Interrupt Controller Virtual Interrupt Priority Mask Register

[ICV_RPR](#): Interrupt Controller Virtual Running Priority Register

[ID_AFR0](#): Auxiliary Feature Register 0

[ID_DFR0](#): Debug Feature Register 0

[ID_DFR1](#): Debug Feature Register 1

[ID_ISAR0](#): Instruction Set Attribute Register 0

[ID_ISAR1](#): Instruction Set Attribute Register 1

[ID_ISAR2](#): Instruction Set Attribute Register 2

[ID_ISAR3](#): Instruction Set Attribute Register 3

[ID_ISAR4](#): Instruction Set Attribute Register 4

[ID_ISAR5](#): Instruction Set Attribute Register 5

[ID_ISAR6](#): Instruction Set Attribute Register 6

[ID_MMFR0](#): Memory Model Feature Register 0

[ID_MMFR1](#): Memory Model Feature Register 1

[ID_MMFR2](#): Memory Model Feature Register 2

[ID_MMFR3](#): Memory Model Feature Register 3

[ID_MMFR4](#): Memory Model Feature Register 4

[ID_MMFR5](#): Memory Model Feature Register 5

[ID_PFR0](#): Processor Feature Register 0

[ID_PFR1](#): Processor Feature Register 1

[ID_PFR2](#): Processor Feature Register 2

[IFAR](#): Instruction Fault Address Register

[IFSR](#): Instruction Fault Status Register

[ISR](#): Interrupt Status Register

[JIDR](#): Jazelle ID Register

[JMCR](#): Jazelle Main Configuration Register

[JOSCR](#): Jazelle OS Control Register

[MAIR0](#): Memory Attribute Indirection Register 0

[MAIR1](#): Memory Attribute Indirection Register 1

[MIDR](#): Main ID Register

[MPIDR](#): Multiprocessor Affinity Register

[MVBAR](#): Monitor Vector Base Address Register

[MVFR0](#): Media and VFP Feature Register 0

[MVFR1](#): Media and VFP Feature Register 1

[MVFR2](#): Media and VFP Feature Register 2

[NMRR](#): Normal Memory Remap Register

[NSACR](#): Non-Secure Access Control Register

[PAR](#): Physical Address Register

[PMCCFILTR](#): Performance Monitors Cycle Count Filter Register

[PMCCNTR](#): Performance Monitors Cycle Count Register

[PMCEID0](#): Performance Monitors Common Event Identification register 0

[PMCEID1](#): Performance Monitors Common Event Identification register 1

[PMCEID2](#): Performance Monitors Common Event Identification register 2

[PMCEID3](#): Performance Monitors Common Event Identification register 3

[PMCNTENCLR](#): Performance Monitors Count Enable Clear register

[PMCNTENSET](#): Performance Monitors Count Enable Set register

[PMCR](#): Performance Monitors Control Register

[PMEVCNTR<n>](#): Performance Monitors Event Count Registers

[PMEVTYPER<n>](#): Performance Monitors Event Type Registers

[PMINTENCLR](#): Performance Monitors Interrupt Enable Clear register

[PMINTENSET](#): Performance Monitors Interrupt Enable Set register

[PMMIR](#): Performance Monitors Machine Identification Register

[PMOVSr](#): Performance Monitors Overflow Flag Status Register

[PMOVSSET](#): Performance Monitors Overflow Flag Status Set register

[PMSELR](#): Performance Monitors Event Counter Selection Register

[PMSWINC](#): Performance Monitors Software Increment register

[PMUSERENR](#): Performance Monitors User Enable Register

[PMXEVCNTR](#): Performance Monitors Selected Event Count Register

[PMXEVTYPER](#): Performance Monitors Selected Event Type Register

[PRRR](#): Primary Region Remap Register

[REVIDR](#): Revision ID Register

[RMR](#): Reset Management Register

[RVBAR](#): Reset Vector Base Address Register

[SCR](#): Secure Configuration Register

[SCTLR](#): System Control Register

[SDCR](#): Secure Debug Control Register

[SDER](#): Secure Debug Enable Register

[SPSR](#): Saved Program Status Register

[SPSR_abt](#): Saved Program Status Register (Abort mode)

[SPSR_fiq](#): Saved Program Status Register (FIQ mode)

[SPSR_hyp](#): Saved Program Status Register (Hyp mode)

[SPSR_irq](#): Saved Program Status Register (IRQ mode)

[SPSR_mon](#): Saved Program Status Register (Monitor mode)

[SPSR_svc](#): Saved Program Status Register (Supervisor mode)

[SPSR_und](#): Saved Program Status Register (Undefined mode)

[TCMTR](#): TCM Type Register

[TLBTR](#): TLB Type Register

[TPIDRPRW](#): PL1 Software Thread ID Register

[TPIDRURO](#): PL0 Read-Only Software Thread ID Register

[TPIDRURW](#): PL0 Read/Write Software Thread ID Register

[TRFCR](#): Trace Filter Control Register

[TTBCR](#): Translation Table Base Control Register

[TTBCR2](#): Translation Table Base Control Register 2

[TTBR0](#): Translation Table Base Register 0

[TTBR1](#): Translation Table Base Register 1

[VBAR](#): Vector Base Address Register

[VDFSR](#): Virtual SError Exception Syndrome Register

[VDISR](#): Virtual Deferred Interrupt Status Register

[VMPIDR](#): Virtualization Multiprocessor ID Register

[VPIDR](#): Virtualization Processor ID Register

[VTCCR](#): Virtualization Translation Control Register

[VTTBR](#): Virtualization Translation Table Base Register

30/09/2021 14:57

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AArch32 System Instructions

[ATS12NSOPR](#): Address Translate Stages 1 and 2 Non-secure Only PL1 Read

[ATS12NSOPW](#): Address Translate Stages 1 and 2 Non-secure Only PL1 Write

[ATS12NSOUR](#): Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read

[ATS12NSOUW](#): Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write

[ATS1CPR](#): Address Translate Stage 1 Current state PL1 Read

[ATS1CPRP](#): Address Translate Stage 1 Current state PL1 Read PAN

[ATS1CPW](#): Address Translate Stage 1 Current state PL1 Write

[ATS1CPWP](#): Address Translate Stage 1 Current state PL1 Write PAN

[ATS1CUR](#): Address Translate Stage 1 Current state Unprivileged Read

[ATS1CUW](#): Address Translate Stage 1 Current state Unprivileged Write

[ATS1HR](#): Address Translate Stage 1 Hyp mode Read

[ATS1HW](#): Address Translate Stage 1 Hyp mode Write

[BPIALL](#): Branch Predictor Invalidate All

[BPIALLIS](#): Branch Predictor Invalidate All, Inner Shareable

[BPIMVA](#): Branch Predictor Invalidate by VA

[CFPRCTX](#): Control Flow Prediction Restriction by Context

[CP15DMB](#): Data Memory Barrier System instruction

[CP15DSB](#): Data Synchronization Barrier System instruction

[CP15ISB](#): Instruction Synchronization Barrier System instruction

[CPPRCTX](#): Cache Prefetch Prediction Restriction by Context

[DCCIMVAC](#): Data Cache line Clean and Invalidate by VA to PoC

[DCCISW](#): Data Cache line Clean and Invalidate by Set/Way

[DCCMVAC](#): Data Cache line Clean by VA to PoC

[DCCMVAU](#): Data Cache line Clean by VA to PoU

[DCCSW](#): Data Cache line Clean by Set/Way

[DCIMVAC](#): Data Cache line Invalidate by VA to PoC

[DCISW](#): Data Cache line Invalidate by Set/Way

[DTLBIALL](#): Data TLB Invalidate All

[DTLBIASID](#): Data TLB Invalidate by ASID match

[DTLBIMVA](#): Data TLB Invalidate by VA

[DVPRCTX](#): Data Value Prediction Restriction by Context

[ICIALLU](#): Instruction Cache Invalidate All to PoU

[ICIALLUIS](#): Instruction Cache Invalidate All to PoU, Inner Shareable

[ICIMVAU](#): Instruction Cache line Invalidate by VA to PoU

[TLBIALL](#): Instruction TLB Invalidate All

[TLBIASID](#): Instruction TLB Invalidate by ASID match

[TLBIMVA](#): Instruction TLB Invalidate by VA

[TLBIALL](#): TLB Invalidate All

[TLBIALLH](#): TLB Invalidate All, Hyp mode

[TLBIALLHIS](#): TLB Invalidate All, Hyp mode, Inner Shareable

[TLBIALLIS](#): TLB Invalidate All, Inner Shareable

[TLBIALLNSNH](#): TLB Invalidate All, Non-Secure Non-Hyp

[TLBIALLNSNHIS](#): TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable

[TLBIASID](#): TLB Invalidate by ASID match

[TLBIASIDIS](#): TLB Invalidate by ASID match, Inner Shareable

[TLBIIPAS2](#): TLB Invalidate by Intermediate Physical Address, Stage 2

[TLBIIPAS2IS](#): TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable

[TLBIIPAS2L](#): TLB Invalidate by Intermediate Physical Address, Stage 2, Last level

[TLBIIPAS2LIS](#): TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable

[TLBIMVA](#): TLB Invalidate by VA

[TLBIMVAA](#): TLB Invalidate by VA, All ASID

[TLBIMVAAIS](#): TLB Invalidate by VA, All ASID, Inner Shareable

[TLBIMVAAL](#): TLB Invalidate by VA, All ASID, Last level

[TLBIMVAALIS](#): TLB Invalidate by VA, All ASID, Last level, Inner Shareable

[TLBIMVAH](#): TLB Invalidate by VA, Hyp mode

[TLBIMVAHIS](#): TLB Invalidate by VA, Hyp mode, Inner Shareable

[TLBIMVAIS](#): TLB Invalidate by VA, Inner Shareable

[TLBIMVAL](#): TLB Invalidate by VA, Last level

[TLBIMVALH](#): TLB Invalidate by VA, Last level, Hyp mode

[TLBIMVALHIS](#): TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable

[TLBIMVALIS](#): TLB Invalidate by VA, Last level, Inner Shareable

30/09/2021 14:57

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ACTLR, Auxiliary Control Register

The ACTLR characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED configuration and control options for execution at EL1 and EL0.

Configuration

AArch32 System register ACTLR bits [31:0] are architecturally mapped to AArch64 System register [ACTLR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ACTLR are UNDEFINED.

Some bits might define global configuration settings, and be common to the Secure and Non-secure instances of the register.

Attributes

ACTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ACTLR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TACR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TAC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return ACTLR_NS;
    else
        return ACTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return ACTLR_NS;
    else
        return ACTLR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return ACTLR_S;
    else
        return ACTLR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TACR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TAC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        ACTLR_NS = R[t];
    else
        ACTLR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        ACTLR_NS = R[t];
    else
        ACTLR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        ACTLR_S = R[t];
    else
        ACTLR_NS = R[t];

```

ACTLR2, Auxiliary Control Register 2

The ACTLR2 characteristics are:

Purpose

Provides additional space to the ACTLR register to hold IMPLEMENTATION DEFINED trap functionality for execution at EL1 and EL0.

Configuration

AArch32 System register ACTLR2 bits [31:0] are architecturally mapped to AArch64 System register [ACTLR_EL1\[63:32\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ACTLR2 are UNDEFINED.

In Armv8.0 and Armv8.1, it is IMPLEMENTATION DEFINED whether this register is implemented, or whether it causes UNDEFINED exceptions when accessed. The implementation of this register can be detected by examining [ID_MMFR4.AC2](#).

From Armv8.2 this register must be implemented.

Attributes

ACTLR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ACTLR2

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TACR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TAC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return ACTLR2_NS;
    else
        return ACTLR2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return ACTLR2_NS;
    else
        return ACTLR2;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return ACTLR2_S;
    else
        return ACTLR2_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TACR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TAC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        ACTLR2_NS = R[t];
    else
        ACTLR2 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        ACTLR2_NS = R[t];
    else
        ACTLR2 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        ACTLR2_S = R[t];
    else
        ACTLR2_NS = R[t];

```

ADFSR, Auxiliary Data Fault Status Register

The ADFSRR characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for Data Abort exceptions taken to EL1 modes, and EL3 modes when EL3 is implemented and is using AArch32.

Configuration

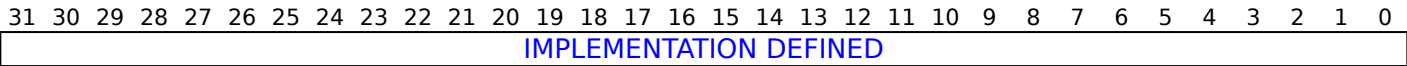
AArch32 System register ADFSRR bits [31:0] are architecturally mapped to AArch64 System register [AFSR0_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ADFSRR are UNDEFINED.

Attributes

ADFSRR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ADFSRR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return ADFSRS_NS;
    else
        return ADFSRS;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return ADFSRS_NS;
    else
        return ADFSRS;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return ADFSRS_S;
    else
        return ADFSRS_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        ADFSRS_NS = R[t];
    else
        ADFSRS = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        ADFSRS_NS = R[t];
    else
        ADFSRS = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        ADFSRS_S = R[t];
    else
        ADFSRS_NS = R[t];

```

AIDR, Auxiliary ID Register

The AIDR characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED identification information.

The value of this register must be used in conjunction with the value of [MIDR](#).

Configuration

AArch32 System register AIDR bits [31:0] are architecturally mapped to AArch64 System register [AIDR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to AIDR are UNDEFINED.

Attributes

AIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing AIDR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b001 | 0b0000 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return AIDR;
elsif PSTATE.EL == EL2 then
    return AIDR;
elsif PSTATE.EL == EL3 then
    return AIDR;

```


AIFSR, Auxiliary Instruction Fault Status Register

The AIFSR characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED fault status information for Prefetch Abort exceptions taken to EL1 modes, and EL3 modes when EL3 is implemented and is using AArch32.

Configuration

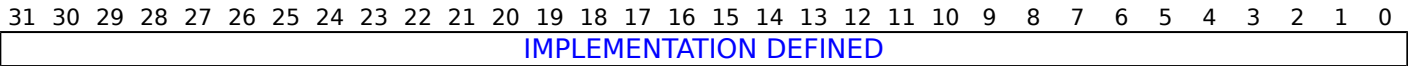
AArch32 System register AIFSR bits [31:0] are architecturally mapped to AArch64 System register [AFSR1_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to AIFSR are UNDEFINED.

Attributes

AIFSR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AIFSR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return AIFSR_NS;
    else
        return AIFSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return AIFSR_NS;
    else
        return AIFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return AIFSR_S;
    else
        return AIFSR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        AIFSR_NS = R[t];
    else
        AIFSR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        AIFSR_NS = R[t];
    else
        AIFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        AIFSR_S = R[t];
    else
        AIFSR_NS = R[t];

```

AMAIRO, Auxiliary Memory Attribute Indirection Register 0

The AMAIRO characteristics are:

Purpose

When using the Long-descriptor format translation tables for stage 1 translations, provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by [MAIRO](#).

Configuration

AArch32 System register AMAIRO bits [31:0] are architecturally mapped to AArch64 System register [AMAIR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to AMAIRO are UNDEFINED.

Attributes

AMAIRO is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

This register is RES0 in the following cases:

- When an implementation does not provide any IMPLEMENTATION DEFINED memory attributes.
- When the Long-descriptor translation table format is not used.

If EL3 is implemented and is using AArch32:

- AMAIRO(S) gives the value for memory accesses from Secure state.
- AMAIRO(NS) gives the value for memory accesses from Non-secure states other than Hyp mode.

Any IMPLEMENTATION DEFINED memory attributes are additional qualifiers for the memory locations and must not change the architected behavior specified by [MAIRO](#) and [MAIR1](#).

In a typical implementation, AMAIRO and [AMAIR1](#) split into eight one-byte fields, corresponding to the MAIRn.Attr<n> fields, but the architecture does not require them to do so.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMAIRO

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| | | | | |
|--------|------|-----|-----|------|
| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1010 | 0b0011 | 0b000 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return AMAIRO_NS;
    else
        return AMAIRO;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return AMAIRO_NS;
    else
        return AMAIRO;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return AMAIRO_S;
    else
        return AMAIRO_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        AMAIRO_NS = R[t];
    else
        AMAIRO = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        AMAIRO_NS = R[t];
    else
        AMAIRO = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            AMAIRO_S = R[t];
        else
            AMAIRO_NS = R[t];

```


AMAIR1, Auxiliary Memory Attribute Indirection Register 1

The AMAIR1 characteristics are:

Purpose

When using the Long-descriptor format translation tables for stage 1 translations, provides IMPLEMENTATION DEFINED memory attributes for the memory regions specified by [MAIR1](#).

Configuration

AArch32 System register AMAIR1 bits [31:0] are architecturally mapped to AArch64 System register [AMAIR_EL1\[63:32\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to AMAIR1 are UNDEFINED.

When EL3 is using AArch32, write access to AMAIR1(S) is disabled when the CP15SDISABLE signal is asserted HIGH.

Attributes

AMAIR1 is a 32-bit register.

Field descriptions

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

This register is RES0 in the following cases:

- When an implementation does not provide any IMPLEMENTATION DEFINED memory attributes.
- When the Long-descriptor translation table format is not used.

If EL3 is implemented and is using AArch32:

- AMAIR1(S) gives the value for memory accesses from Secure state.
- AMAIR1(NS) gives the value for memory accesses from Non-secure states other than Hyp mode.

Any IMPLEMENTATION DEFINED memory attributes are additional qualifiers for the memory locations and must not change the architected behavior specified by [MAIR0](#) and [MAIR1](#).

In a typical implementation, [AMAIR0](#) and AMAIR1 split into eight one-byte fields, corresponding to the MAIRn.Attr<n> fields, but the architecture does not require them to do so.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMAIR1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1010 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return AMAIR1_NS;
    else
        return AMAIR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return AMAIR1_NS;
    else
        return AMAIR1;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return AMAIR1_S;
    else
        return AMAIR1_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1010 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        AMAIR1_NS = R[t];
    else
        AMAIR1 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        AMAIR1_NS = R[t];
    else
        AMAIR1 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            AMAIR1_S = R[t];
        else
            AMAIR1_NS = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCFGR, Activity Monitors Configuration Register

The AMCFGR characteristics are:

Purpose

Global configuration register for the activity monitors.

Provides information on supported features, the number of counter groups implemented, the total number of activity monitor event counters implemented, and the size of the counters. AMCFGR is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch32 System register AMCFGR bits [31:0] are architecturally mapped to AArch64 System register [AMCFGR_EL0\[31:0\]](#).

AArch32 System register AMCFGR bits [31:0] are architecturally mapped to External register [AMCFGR\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCFGR are UNDEFINED.

Attributes

AMCFGR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|----|----|------|----|------|-----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NCG | | | | RES0 | | HDBG | RAZ | | | | | | | SIZE | | | | N | | | | | | | | | | | | | |

NCG, bits [31:28]

Defines the number of counter groups.

The number of implemented counter groups is [AMCFGR.NCG + 1].

If the number of implemented auxiliary activity monitor event counters is zero, this field has a value of 0b0000. Otherwise, this field has a value of 0b0001.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [27:25]

Reserved, RES0.

HDBG, bit [24]

Halt-on-debug supported.

This feature must be supported, and so this bit is 0b1.

| HDBG | Meaning |
|------|--|
| 0b0 | AMCR.HDBG is RES0. |
| 0b1 | AMCR.HDBG is read/write. |

Access to this field is **RO**.

Bits [23:14]

Reserved, RAZ.

SIZE, bits [13:8]

Defines the size of activity monitor event counters.

The size of the activity monitor event counters implemented by the Activity Monitors Extension is [AMCFGR.SIZE + 1].

The counters are 64-bit.

Note

Software also uses this field to determine the spacing of counters in the memory-map. The counters are at doubleword-aligned addresses.

Reads as 0b111111.

Access to this field is **RO**.

N, bits [7:0]

Defines the number of activity monitor event counters.

The total number of counters implemented in all groups by the Activity Monitors Extension is [AMCFGR.N + 1].

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMCFGR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCFGR;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCFGR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCFGR;
elsif PSTATE.EL == EL3 then
    return AMCFGR;

```


AMCGCR, Activity Monitors Counter Group Configuration Register

The AMCGCR characteristics are:

Purpose

Provides information on the number of activity monitor event counters implemented within each counter group.

Configuration

AArch32 System register AMCGCR bits [31:0] are architecturally mapped to AArch64 System register [AMCGCR_EL0\[31:0\]](#).

AArch32 System register AMCGCR bits [31:0] are architecturally mapped to External register [AMCGCR\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCGCR are UNDEFINED.

Attributes

AMCGCR is a 32-bit register.

Field descriptions

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|-------|---|---|---|---|---|---|---|
| RES0 | | | | | | | | | | | | | | | | CG1NC | | | | | | | | CG0NC | | | | | | | |

Bits [31:16]

Reserved, RES0.

CG1NC, bits [15:8]

Counter Group 1 Number of Counters. The number of counters in the auxiliary counter group.

In an implementation that includes FEAT_AMUv1, the permitted range of values is 0 to 16.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CG0NC, bits [7:0]

Counter Group 0 Number of Counters. The number of counters in the architected counter group.

Reads as 0x04.

Access to this field is **RO**.

Accessing AMCGCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCGCR;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCGCR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCGCR;
elsif PSTATE.EL == EL3 then
    return AMCGCR;

```


AMCNTENCLR0, Activity Monitors Count Enable Clear Register 0

The AMCNTENCLR0 characteristics are:

Purpose

Disable control bits for the architected activity monitors event counters, [AMEVCNTR0<n>](#).

Configuration

AArch32 System register AMCNTENCLR0 bits [31:0] are architecturally mapped to AArch64 System register [AMCNTENCLR0_ELO\[31:0\]](#).

AArch32 System register AMCNTENCLR0 bits [31:0] are architecturally mapped to External register [AMCNTENCLR0\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR0 are UNDEFINED.

Attributes

AMCNTENCLR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|----|----|----|----|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | RAZ/WI | | | | | | | | | | P3 | P2 | P1 | P0 | | |

Bits [31:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n>, bit [n], for n = 3 to 0

Activity monitor event counter disable bit for [AMEVCNTR0<n>](#).

Note

[AMCGCR.CG0NC](#) identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

| P<n> | Meaning |
|-------------------|--|
| 0b0 | When read, means that AMEVCNTR0<n> is disabled. When written, has no effect. |
| 0b1 | When read, means that AMEVCNTR0<n> is enabled. When written, disables AMEVCNTR0<n> . |

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMCNTENCLR0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b100 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENCLR0;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENCLR0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENCLR0;
elsif PSTATE.EL == EL3 then
    return AMCNTENCLR0;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b100 |

```

if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENCLR0 = R[t];
else
    UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENCLR1, Activity Monitors Count Enable Clear Register 1

The AMCNTENCLR1 characteristics are:

Purpose

Disable control bits for the auxiliary activity monitors event counters, [AMEVCNTR1<n>](#).

Configuration

AArch32 System register AMCNTENCLR1 bits [31:0] are architecturally mapped to AArch64 System register [AMCNTENCLR1_ELO\[31:0\]](#).

AArch32 System register AMCNTENCLR1 bits [31:0] are architecturally mapped to External register [AMCNTENCLR1\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR1 are UNDEFINED.

Attributes

AMCNTENCLR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

Bits [31:16]

Reserved, RES0.

P<n>, bit [n], for n = 15 to 0

Activity monitor event counter disable bit for [AMEVCNTR1<n>](#).

When N is less than 16, bits [15:N] are RAZ/WI, where N is the value in [AMCGCR.CG1NC](#).

Possible values of each bit are:

| P<n> | Meaning |
|------|--|
| 0b0 | When read, means that AMEVCNTR1<n> is disabled. When written, has no effect. |
| 0b1 | When read, means that AMEVCNTR1<n> is enabled. When written, disables AMEVCNTR1<n> . |

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMCNTENCLR1

If the number of auxiliary activity monitor event counters implemented is zero, reads and writes of AMCNTENCLR1 are UNDEFINED.

Note

The number of auxiliary activity monitor event counters implemented is zero exactly when [AMCFGR](#).NCG == 0b0000.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENCLR1;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENCLR1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENCLR1;
elsif PSTATE.EL == EL3 then
    return AMCNTENCLR1;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENCLR1 = R[t];
else
    UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENSET0, Activity Monitors Count Enable Set Register 0

The AMCNTENSET0 characteristics are:

Purpose

Enable control bits for the architected activity monitors event counters, [AMEVCNTR0<n>](#).

Configuration

AArch32 System register AMCNTENSET0 bits [31:0] are architecturally mapped to AArch64 System register [AMCNTENSET0_ELO\[31:0\]](#).

AArch32 System register AMCNTENSET0 bits [31:0] are architecturally mapped to External register [AMCNTENSET0\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET0 are UNDEFINED.

Attributes

AMCNTENSET0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|----|----|----|----|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | RAZ/WI | | | | | | | | P3 | P2 | P1 | P0 | | | | |

Bits [31:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n>, bit [n], for n = 3 to 0

Activity monitor event counter enable bit for [AMEVCNTR0<n>](#).

Note

[AMCGCR.CG0NC](#) identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

| P<n> | Meaning |
|-------------------|---|
| 0b0 | When read, means that AMEVCNTR0<n> is disabled. When written, has no effect. |
| 0b1 | When read, means that AMEVCNTR0<n> is enabled. When written, enables AMEVCNTR0<n> . |

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMCNTENSET0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b101 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEN == '1') && HAFGRTR_EL2.AMCNTEN0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENSET0;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENSET0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENSET0;
elsif PSTATE.EL == EL3 then
    return AMCNTENSET0;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b101 |

```

if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENSET0 = R[t];
else
    UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENSET1, Activity Monitors Count Enable Set Register 1

The AMCNTENSET1 characteristics are:

Purpose

Enable control bits for the auxiliary activity monitors event counters, [AMEVCNTR1<n>](#).

Configuration

AArch32 System register AMCNTENSET1 bits [31:0] are architecturally mapped to AArch64 System register [AMCNTENSET1_ELO\[31:0\]](#).

AArch32 System register AMCNTENSET1 bits [31:0] are architecturally mapped to External register [AMCNTENSET1\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET1 are UNDEFINED.

Attributes

AMCNTENSET1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

Bits [31:16]

Reserved, RES0.

P<n>, bit [n], for n = 15 to 0

Activity monitor event counter enable bit for [AMEVCNTR1<n>](#).

When N is less than 16, bits [15:N] are RAZ/WI, where N is the value in [AMCGCR.CG1NC](#).

Possible values of each bit are:

| P<n> | Meaning |
|------|---|
| 0b0 | When read, means that AMEVCNTR1<n> is disabled. When written, has no effect. |
| 0b1 | When read, means that AMEVCNTR1<n> is enabled. When written, enables AMEVCNTR1<n> . |

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMCNTENSET1

If the number of auxiliary activity monitor event counters implemented is zero, reads and writes of AMCNTENSET1 are UNDEFINED.

Note

The number of auxiliary activity monitor counters implemented is zero when [AMCFGR.NCG](#) == 0b0000.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMCNTEN1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENSET1;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENSET1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCNTENSET1;
elsif PSTATE.EL == EL3 then
    return AMCNTENSET1;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
    AMCNTENSET1 = R[t];
else
    UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCR, Activity Monitors Control Register

The AMCR characteristics are:

Purpose

Global control register for the activity monitors implementation. AMCR is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch32 System register AMCR bits [31:0] are architecturally mapped to AArch64 System register [AMCR_EL0\[31:0\]](#).

AArch32 System register AMCR bits [31:0] are architecturally mapped to External register [AMCR\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCR are UNDEFINED.

Attributes

AMCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|------|----|----|----|----|----|------|------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | CG1RZ | RES0 | | | | | | HDBG | RES0 | | | | | | | | | |

Bits [31:18]

Reserved, RES0.

CG1RZ, bit [17]

When FEAT_AMUv1p1 is implemented:

Counter Group 1 Read Zero.

| CG1RZ | Meaning |
|-------|--|
| 0b0 | System register reads of AMEVCNTR1<n> return the event count at all implemented and enabled Exception levels. |
| 0b1 | If the current Exception level is the highest implemented Exception level, system register reads of AMEVCNTR1<n> return the event count. Otherwise, reads of AMEVCNTR1<n> return a zero value. |

Note

Reads from the memory-mapped view are unaffected by this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:11]

Reserved, RES0.

HDBG, bit [10]

This bit controls whether activity monitor counting is halted when the PE is halted in Debug state.

| HDBG | Meaning |
|------|--|
| 0b0 | Activity monitors do not halt counting when the PE is halted in Debug state. |
| 0b1 | Activity monitors halt counting when the PE is halted in Debug state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [9:0]

Reserved, RES0.

Accessing AMCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCR;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMCR;
elsif PSTATE.EL == EL3 then
    return AMCR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) then
    AMCR = R[t];
else
    UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVCNTR0<n>, Activity Monitors Event Counter Registers 0, n = 0 - 3

The AMEVCNTR0<n> characteristics are:

Purpose

Provides access to the architected activity monitor event counters.

Configuration

AArch32 System register AMEVCNTR0<n> bits [63:0] are architecturally mapped to AArch64 System register [AMEVCNTR0<n>_EL0\[63:0\]](#).

AArch32 System register AMEVCNTR0<n> bits [63:0] are architecturally mapped to External register [AMEVCNTR0<n>\[63:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR0<n> are UNDEFINED.

Attributes

AMEVCNTR0<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | ACNT | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ACNT | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ACNT, bits [63:0]

Architected activity monitor event counter n.

Value of architected activity monitor event counter n, where n is the number of this register and is a number from 0 to 3.

If FEAT_AMUv1p1 is implemented, [HCR_EL2](#).AMVOFFEN is 1, [SCR_EL3](#).AMVOFFEN is 1, [HCR_EL2](#).{E2H, TGE} is not {1,1}, and EL2 is using AArch64 and is implemented in the current Security state, access to these registers at EL0 or EL1 return (PCount<63:0> - [AMEVCNTVOFF0<n>_EL2](#)<63:0>).

PCount is the physical count returned when AMEVCNTR0<n> is read from EL2 or EL3.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMEVCNTR0<n>

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVCNTR0<n> are UNDEFINED.

Note

[AMCGCR.CG0NC](#) identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|------------|------------|
| 0b1111 | 0b000:n[3] | 0b0:n[2:0] |

```

if CRm == '0000' then
    if PSTATE.EL == EL0 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            else
                AArch64.AArch32SystemAccessTrap(EL1, 0x04);
            elsif ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
                if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
                elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                    AArch32.TakeHypTrapException(0x00);
                else
                    UNDEFINED;
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T0 ==
'1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
                AArch32.TakeHypTrapException(0x04);
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
                AArch32.TakeHypTrapException(0x04);
            elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3)
|| SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVCNTR0<n>_EL0 == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x04);
                else
                    return AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)];
            elsif PSTATE.EL == EL1 then
                if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                    UNDEFINED;
                elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
                    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
                elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
                    AArch32.TakeHypTrapException(0x04);
                elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
                    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
                elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
                    AArch32.TakeHypTrapException(0x04);
                elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
                    else
                        return AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)];
            elsif PSTATE.EL == EL2 then
                if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                    UNDEFINED;
                elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
                    else
                        return AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)];
            elsif PSTATE.EL == EL3 then
                return AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)];
        else

```

UNDEFINED;

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|------------|------------|
| 0b1111 | 0b000:n[3] | 0b0:n[2:0] |

```

if CRm == '0000' then
    if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif IsHighestEL(PSTATE.EL) then
        AMEVCNTR0[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
    else
        UNDEFINED;
else
    UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVCNTR1<n>, Activity Monitors Event Counter Registers 1, n = 0 - 15

The AMEVCNTR1<n> characteristics are:

Purpose

Provides access to the auxiliary activity monitor event counters.

Configuration

AArch32 System register AMEVCNTR1<n> bits [63:0] are architecturally mapped to AArch64 System register [AMEVCNTR1<n>_EL0\[63:0\]](#).

AArch32 System register AMEVCNTR1<n> bits [63:0] are architecturally mapped to External register [AMEVCNTR1<n>\[63:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR1<n> are UNDEFINED.

Attributes

AMEVCNTR1<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | ACNT | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ACNT | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ACNT, bits [63:0]

Auxiliary activity monitor event counter n.

Value of auxiliary activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

If FEAT_AMUv1p1 is implemented, [HCR_EL2](#).AMVOFFEN is 1, [SCR_EL3](#).AMVOFFEN is 1, [HCR_EL2](#).{E2H, TGE} is not {1,1}, EL2 is using AArch64 and is implemented in the current Security state, and [AMCR_EL0](#).CG1RZ is 0, reads to these registers at EL0 or EL1 return (PCount<63:0> - [AMEVCNTVOFF1<n>_EL2](#)<63:0>).

PCount is the physical count returned when AMEVCNTR1<n> is read from EL2 or EL3.

If the counter is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMEVCNTR1<n>

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVCNTR1<n> are UNDEFINED.

Note

[AMCGCR](#).CG1NC identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|------------|------------|
| 0b1111 | 0b010:n[3] | 0b0:n[2:0] |

```

if CRm == '0100' then
    if PSTATE.EL == EL0 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elseif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            else
                AArch64.AArch32SystemAccessTrap(EL1, 0x04);
            elseif ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
                if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
                elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                    AArch32.TakeHypTrapException(0x00);
                else
                    UNDEFINED;
            elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elseif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
                AArch32.TakeHypTrapException(0x04);
            elseif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3)
|| SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVCNTR1<n>_EL0 == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x04);
            elseif HaveAArch64() && AMCR_EL0.CG1RZ == '1' then
                return Zeros();
            elseif !HaveAArch64() && AMCR.CG1RZ == '1' then
                return Zeros();
            else
                return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
        elseif PSTATE.EL == EL1 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                UNDEFINED;
            elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elseif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
                AArch32.TakeHypTrapException(0x04);
            elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x04);
            elseif !IsHighestEL(PSTATE.EL) && HaveAArch64() && AMCR_EL0.CG1RZ == '1' then
                return Zeros();
            elseif !IsHighestEL(PSTATE.EL) && !HaveAArch64() && AMCR.CG1RZ == '1' then
                return Zeros();
            else
                return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
        elseif PSTATE.EL == EL2 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                UNDEFINED;
            elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x04);
            elseif !IsHighestEL(PSTATE.EL) && HaveAArch64() && AMCR_EL0.CG1RZ == '1' then
                return Zeros();
            elseif !IsHighestEL(PSTATE.EL) && !HaveAArch64() && AMCR.CG1RZ == '1' then
                return Zeros();
            else
                return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
        elseif PSTATE.EL == EL3 then

```

```

        return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
elseif CRm == '0101' then
    if PSTATE.EL == EL0 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elseif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            else
                AArch64.AArch32SystemAccessTrap(EL1, 0x04);
            elseif ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
                if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
                elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                    AArch32.TakeHypTrapException(0x00);
                else
                    UNDEFINED;
            elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T5 ==
'1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
                AArch32.TakeHypTrapException(0x04);
            elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elseif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
                AArch32.TakeHypTrapException(0x04);
            elseif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3)
|| SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVCNTR1<n>_EL0 == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x04);
            elseif HaveAArch64() && AMCR_EL0.CG1RZ == '1' then
                return Zeros();
            elseif !HaveAArch64() && AMCR.CG1RZ == '1' then
                return Zeros();
            else
                return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
        elseif PSTATE.EL == EL1 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                UNDEFINED;
            elseif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
                AArch32.TakeHypTrapException(0x04);
            elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elseif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
                AArch32.TakeHypTrapException(0x04);
            elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x04);
            elseif !IsHighestEL(PSTATE.EL) && HaveAArch64() && AMCR_EL0.CG1RZ == '1' then
                return Zeros();
            elseif !IsHighestEL(PSTATE.EL) && !HaveAArch64() && AMCR.CG1RZ == '1' then
                return Zeros();
            else
                return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
        elseif PSTATE.EL == EL2 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                UNDEFINED;
            elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                if Halted() && EDSCR.SDD == '1' then

```

```

        UNDEFINED;
    else
        AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    elsif !IsHighestEL(PSTATE.EL) && HaveAArch64() && AMCR_EL0.CG1RZ == '1' then
        return Zeros();
    elsif !IsHighestEL(PSTATE.EL) && !HaveAArch64() && AMCR.CG1RZ == '1' then
        return Zeros();
    else
        return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
    elsif PSTATE.EL == EL3 then
        return AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)];
    else
        UNDEFINED;

```

MCCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|------------|------------|
| 0b1111 | 0b010:n[3] | 0b0:n[2:0] |

```

if CRm == '0100' then
    if IsHighestEL(PSTATE.EL) then
        AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
    else
        UNDEFINED;
elsif CRm == '0101' then
    if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif IsHighestEL(PSTATE.EL) then
        AMEVCNTR1[UInt(CRm<0>:opc1<2:0>)] = R[t2]:R[t];
    else
        UNDEFINED;
else
    UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVTYPER0<n>, Activity Monitors Event Type Registers 0, n = 0 - 3

The AMEVTYPER0<n> characteristics are:

Purpose

Provides information on the events that an architected activity monitor event counter [AMEVCNTR0<n>](#) counts.

Configuration

AArch32 System register AMEVTYPER0<n> bits [31:0] are architecturally mapped to AArch64 System register [AMEVTYPER0<n>_EL0\[31:0\]](#).

AArch32 System register AMEVTYPER0<n> bits [31:0] are architecturally mapped to External register [AMEVTYPER0<n>\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER0<n> are UNDEFINED.

Attributes

AMEVTYPER0<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | evtCount | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the architected activity monitor event counter [AMEVCNTR0<n>](#). The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

| evtCount | Meaning | Applies when |
|----------|----------------------------|--------------|
| 0x0011 | Processor frequency cycles | When n == 0 |
| 0x4004 | Constant frequency cycles | When n == 1 |
| 0x0008 | Instructions retired | When n == 2 |
| 0x4005 | Memory stall cycles | When n == 3 |

Accessing AMEVTYPER0<n>

If <n> is greater than or equal to the number of architected activity monitor event counters, reads and writes of AMEVTYPER0<n> are UNDEFINED.

Note

[AMCGCR.CG0NC](#) identifies the number of architected activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1101 | 0b011:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elseif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMEVTYPER0[UInt(CRm<0>:opc2<2:0>)];
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMEVTYPER0[UInt(CRm<0>:opc2<2:0>)];
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMEVTYPER0[UInt(CRm<0>:opc2<2:0>)];
elseif PSTATE.EL == EL3 then
    return AMEVTYPER0[UInt(CRm<0>:opc2<2:0>)];

```


AMEVTYPER1<n>, Activity Monitors Event Type Registers 1, n = 0 - 15

The AMEVTYPER1<n> characteristics are:

Purpose

Provides information on the events that an auxiliary activity monitor event counter [AMEVCNTR1<n>](#) counts.

Configuration

AArch32 System register AMEVTYPER1<n> bits [31:0] are architecturally mapped to AArch64 System register [AMEVTYPER1<n>_EL0\[31:0\]](#).

AArch32 System register AMEVTYPER1<n> bits [31:0] are architecturally mapped to External register [AMEVTYPER1<n>\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER1<n> are UNDEFINED.

Attributes

AMEVTYPER1<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | evtCount | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the auxiliary activity monitor event counter [AMEVCNTR1<n>](#).

It is IMPLEMENTATION DEFINED what values are supported by each counter.

If software writes a value to this field which is not supported by the corresponding counter [AMEVCNTR1<n>](#), then:

- It is UNPREDICTABLE which event will be counted.
- The value read back is UNKNOWN.

The event counted by [AMEVCNTR1<n>](#) might be fixed at implementation. In this case, the field is read-only and writes are UNDEFINED.

If the corresponding counter [AMEVCNTR1<n>](#) is enabled, writes to this register have UNPREDICTABLE results.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMEVTYPER1<n>

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads and writes of AMEVTYPER1<n> are UNDEFINED.

Note

[AMCGCR.CG1NC](#) identifies the number of auxiliary activity monitor event counters.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|------------|--------|
| 0b1111 | 0b000 | 0b1101 | 0b111:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && AMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HAFGRTR_EL2.AMEVTYPER1<n>_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL3 then
    return AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|------------|--------|
| 0b1111 | 0b000 | 0b1101 | 0b111.n[3] | n[2:0] |

```

if PSTATE.EL == EL1 && EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
    AArch32.TakeHypTrapException(0x03);
elsif IsHighestEL(PSTATE.EL) && !boolean IMPLEMENTATION_DEFINED "AMEVCNTR1<n> is fixed" then
    AMEVTYPER1[UInt(CRm<0>:opc2<2:0>)] = R[t];
else
    UNDEFINED;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMUSERENR, Activity Monitors User Enable Register

The AMUSERENR characteristics are:

Purpose

Global user enable register for the activity monitors. Enables or disables EL0 access to the activity monitors. AMUSERENR is applicable to both the architected and the auxiliary counter groups.

Configuration

AArch32 System register AMUSERENR bits [31:0] are architecturally mapped to AArch64 System register [AMUSERENR_ELO\[31:0\]](#).

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMUSERENR are UNDEFINED.

Attributes

AMUSERENR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | EN |

Bits [31:1]

Reserved, RES0.

EN, bit [0]

Traps EL0 accesses to the activity monitors registers to EL1.

| EN | Meaning |
|-----|--|
| 0b0 | EL0 accesses to the activity monitors registers are trapped to EL1. |
| 0b1 | This control does not cause any instructions to be trapped. Software can access all activity monitor registers at EL0. |

Note

- AMUSERENR can always be read at EL0 and is not governed by this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMUSERENR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return AMUSERENR;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            else
                return AMUSERENR;
        elsif PSTATE.EL == EL2 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                UNDEFINED;
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
                else
                    return AMUSERENR;
        elsif PSTATE.EL == EL3 then
            return AMUSERENR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TAM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TAM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            AMUSERENR = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TAM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            else
                AMUSERENR = R[t];
    elsif PSTATE.EL == EL3 then
        AMUSERENR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

APSR, Application Program Status Register

The APSR characteristics are:

Purpose

Hold program status and control information.

Note

Some of the fields in this register are permitted to return the value of the PSTATE field on a read. This is an exception to the general rule that an UNKNOWN field must not return information that cannot be obtained, at the current Privilege level, by an architected mechanism.

For more information see 'The Application Program Status Register, APSR'.

Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to APSR are UNDEFINED.

Attributes

APSR is a 32-bit register.

Field descriptions

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|----|----|----|----|----|------|----|-----|----|------|----|----|------|----|----|----|----|----|----|----|------|--------|---|---|---|---|---|---|---|---|---|---|
| N | Z | C | V | Q | RES0 | | PAN | | RES0 | GE | | RES0 | | | | E | A | I | F | RES0 | M[4:0] | | | | | | | | | | |

N, bit [31]

Negative condition flag. Set to bit[31] of the result of the last flag-setting instruction. If the result is regarded as a two's complement signed integer, then N is set to 1 if the result was negative, and N is set to 0 if the result was positive or zero.

Z, bit [30]

Zero condition flag. Set to 1 if the result of the last flag-setting instruction was zero, and to 0 otherwise. A result of zero often indicates an equal result from a comparison.

C, bit [29]

Carry condition flag. Set to 1 if the last flag-setting instruction resulted in a carry condition, for example an unsigned overflow on an addition.

V, bit [28]

Overflow condition flag. Set to 1 if the last flag-setting instruction resulted in an overflow condition, for example a signed overflow on an addition.

Q, bit [27]

Cumulative saturation bit. Set to 1 to indicate that overflow or saturation occurred in some instructions.

Bits [26:23]

Reserved, RES0.

PAN, bit [22]**When FEAT_PAN is implemented:**

Privileged Access Never. This field is UNKNOWN, but is permitted to return the value of PSTATE.PAN field. On writes, this field is treated as Do-Not-Modify.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [21:20]

Reserved, RES0.

GE, bits [19:16]

Greater than or Equal flags, for parallel addition and subtraction.

Bits [15:10]

Reserved, RES0.

E, bit [9]

Endianness. This field is UNKNOWN, but is permitted to return the value of PSTATE.E field. On writes, this field is treated as Do-Not-Modify.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. This field is UNKNOWN, but is permitted to return the value of PSTATE.A field. On writes, this field is treated as Do-Not-Modify.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. This field is UNKNOWN, but is permitted to return the value of PSTATE.I field. On writes, this field is treated as Do-Not-Modify.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. This field is UNKNOWN, but is permitted to return the value of PSTATE.F field. On writes, this field is treated as Do-Not-Modify.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

M[4:0], bits [4:0]

Mode. This field is UNKNOWN, but is permitted to return the value of PSTATE.M[4:0] field. On writes, this field is treated as Do-Not-Modify.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing APSR

APSR can be read using the MRS instruction and written using the MSR (register) or MSR (immediate) instructions.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS12NSOPR, Address Translate Stages 1 and 2 Non-secure Only PL1 Read

The ATS12NSOPR characteristics are:

Purpose

Performs stage 1 and 2 address translations as defined for PL1 and the Non-secure state, with permissions as if reading from the given virtual address.

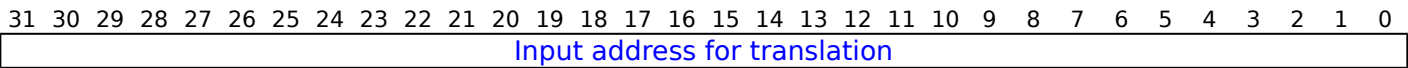
Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ATS12NSOPR are UNDEFINED.

Attributes

ATS12NSOPR is a 32-bit System instruction.

Field descriptions



Bits [31:0]

Input address for translation. The resulting address can be read from the [PAR](#).

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

Executing the ATS12NSOPR instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_12, EL1, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_12, EL1, ATAccess_Read);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS12NSOPW, Address Translate Stages 1 and 2 Non-secure Only PL1 Write

The ATS12NSOPW characteristics are:

Purpose

Performs stage 1 and 2 address translations as defined for PL1 and the Non-secure state, with permissions as if writing to the given virtual address.

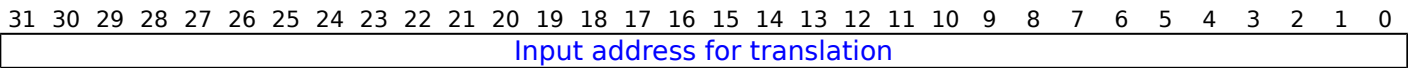
Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ATS12NSOPW are UNDEFINED.

Attributes

ATS12NSOPW is a 32-bit System instruction.

Field descriptions



Bits [31:0]

Input address for translation. The resulting address can be read from the [PAR](#).

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

Executing the ATS12NSOPW instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_12, EL1, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_12, EL1, ATAccess_Write);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS12NSOUR, Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read

The ATS12NSOUR characteristics are:

Purpose

Performs stage 1 and 2 address translations as defined for PL0 and the Non-secure state, with permissions as if reading from the given virtual address.

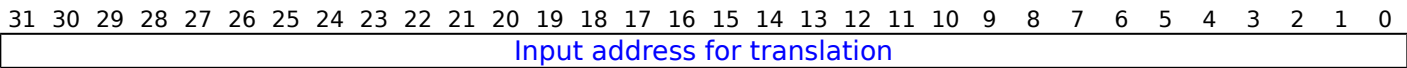
Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ATS12NSOUR are UNDEFINED.

Attributes

ATS12NSOUR is a 32-bit System instruction.

Field descriptions



Bits [31:0]

Input address for translation. The resulting address can be read from the [PAR](#).

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

Executing the ATS12NSOUR instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_12, EL0, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_12, EL0, ATAccess_Read);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS12NSOUW, Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write

The ATS12NSOUW characteristics are:

Purpose

Performs stage 1 and 2 address translations as defined for PL0 and the Non-secure state, with permissions as if writing to the given virtual address.

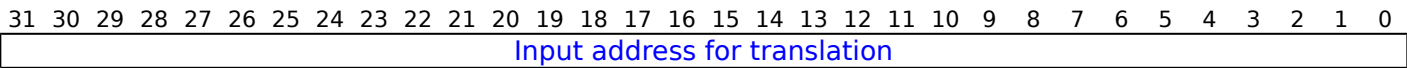
Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ATS12NSOUW are UNDEFINED.

Attributes

ATS12NSOUW is a 32-bit System instruction.

Field descriptions



Bits [31:0]

Input address for translation. The resulting address can be read from the [PAR](#).

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the stage 2 translation.

Executing the ATS12NSOUW instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_12, EL0, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_12, EL0, ATAccess_Write);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1CPR, Address Translate Stage 1 Current state PL1 Read

The ATS1CPR characteristics are:

Purpose

Performs stage 1 address translation as defined for PL1 and the current Security state, with permissions as if reading from the given virtual address.

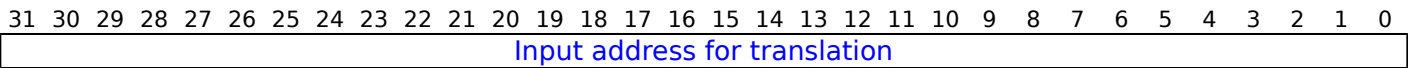
Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ATS1CPR are UNDEFINED.

Attributes

ATS1CPR is a 32-bit System instruction.

Field descriptions



Bits [31:0]

Input address for translation. The resulting address can be read from the [PAR](#).

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing the ATS1CPR instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_Read);
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        AArch32.AT(R[t], TranslationStage_1, EL3, ATAccess_Read);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_Read);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1CPRP, Address Translate Stage 1 Current state PL1 Read PAN

The ATS1CPRP characteristics are:

Purpose

Performs a stage 1 address translation at PL1 and in the current Security state, where the value of PSTATE.PAN determines if a read from a location will generate a Permission fault for a privileged access.

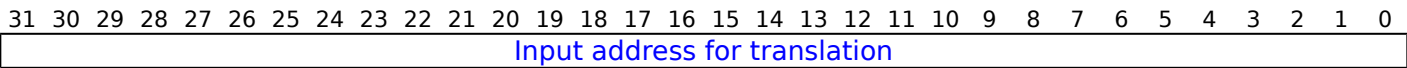
Configuration

This instruction is present only when AArch32 is supported and FEAT_PAN2 is implemented. Otherwise, direct accesses to ATS1CPRP are UNDEFINED.

Attributes

ATS1CPRP is a 32-bit System instruction.

Field descriptions



Bits [31:0]

Input address for translation. The resulting address can be read from the [PAR](#).

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing the ATS1CPRP instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_ReadPAN);
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_ReadPAN);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        AArch32.AT(R[t], TranslationStage_1, EL3, ATAccess_ReadPAN);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_ReadPAN);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1CPW, Address Translate Stage 1 Current state PL1 Write

The ATS1CPW characteristics are:

Purpose

Performs stage 1 address translation as defined for PL1 and the current Security state, with permissions as if writing to the given virtual address.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ATS1CPW are UNDEFINED.

Attributes

ATS1CPW is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Input address for translation. The resulting address can be read from the [PAR](#).

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing the ATS1CPW instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_Write);
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        AArch32.AT(R[t], TranslationStage_1, EL3, ATAccess_Write);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_Write);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1CPWP, Address Translate Stage 1 Current state PL1 Write PAN

The ATS1CPWP characteristics are:

Purpose

Performs a stage 1 address translation at PL1 and in the current Security state, where the value of PSTATE.PAN determines if a write to the location will generate a Permission fault for a privileged access.

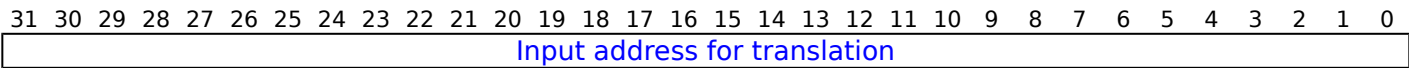
Configuration

This instruction is present only when AArch32 is supported and FEAT_PAN2 is implemented. Otherwise, direct accesses to ATS1CPWP are UNDEFINED.

Attributes

ATS1CPWP is a 32-bit System instruction.

Field descriptions



Bits [31:0]

Input address for translation. The resulting address can be read from the [PAR](#).

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing the ATS1CPWP instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_WritePAN);
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_WritePAN);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        AArch32.AT(R[t], TranslationStage_1, EL3, ATAccess_WritePAN);
    else
        AArch32.AT(R[t], TranslationStage_1, EL1, ATAccess_WritePAN);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1CUR, Address Translate Stage 1 Current state Unprivileged Read

The ATS1CUR characteristics are:

Purpose

Performs stage 1 address translation as defined for PL0 and the current Security state, with permissions as if reading from the given virtual address.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ATS1CUR are UNDEFINED.

Attributes

ATS1CUR is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Input address for translation. The resulting address can be read from the [PAR](#).

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing the ATS1CUR instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.AT(R[t], TranslationStage_1, EL0, ATAccess_Read);
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL0, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_1, EL0, ATAccess_Read);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1CUW, Address Translate Stage 1 Current state Unprivileged Write

The ATS1CUW characteristics are:

Purpose

Performs stage 1 address translation as defined for PL0 and the current Security state, with permissions as if writing to the given virtual address.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ATS1CUW are UNDEFINED.

Attributes

ATS1CUW is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Input address for translation. The resulting address can be read from the [PAR](#).

This System instruction takes a VA as input. If EL2 is implemented and enabled in the current Security state, the resulting address is the IPA that is the output address of the stage 1 translation. Otherwise, the resulting address is a PA.

Executing the ATS1CUW instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.AT(R[t], TranslationStage_1, EL0, ATAccess_Write);
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL0, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_1, EL0, ATAccess_Write);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1HR, Address Translate Stage 1 Hyp mode Read

The ATS1HR characteristics are:

Purpose

Performs stage 1 address translation as defined for PL2 and the Non-secure state, with permissions as if reading from the given virtual address.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ATS1HR are UNDEFINED.

Attributes

ATS1HR is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Input address for translation. The resulting address can be read from the [PAR](#).

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the translation.

Executing the ATS1HR instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0111 | 0b1000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL2, ATAccess_Read);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_1, EL2, ATAccess_Read);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ATS1HW, Address Translate Stage 1 Hyp mode Write

The ATS1HW characteristics are:

Purpose

Performs stage 1 address translation as defined for PL2 and the Non-secure state, with permissions as if writing to the given virtual address.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ATS1HW are UNDEFINED.

Attributes

ATS1HW is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Input address for translation | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Input address for translation. The resulting address can be read from the [PAR](#).

This System instruction takes a VA as input. The resulting address is the PA that is the output address of the translation.

Executing the ATS1HW instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0111 | 0b1000 | 0b001 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.AT(R[t], TranslationStage_1, EL2, ATAccess_Write);
elsif PSTATE.EL == EL3 then
    AArch32.AT(R[t], TranslationStage_1, EL2, ATAccess_Write);
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

BPIALL, Branch Predictor Invalidate All

The BPIALL characteristics are:

Purpose

Invalidate all entries from branch predictors.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to BPIALL are UNDEFINED.

In an implementation where the branch predictors are architecturally invisible, this instruction can execute as a NOP.

Attributes

BPIALL is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the BPIALL instruction

The PE ignores the value of <Rt>. Software does not have to write a value to this register before issuing this instruction.

When [HCR.FB](#) is 1, at Non-secure EL1 this instruction executes as a [BPIALLIS](#).

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b0101 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
        BPIALLIS();
    else
        BPIALL();
elsif PSTATE.EL == EL2 then
    BPIALL();
elsif PSTATE.EL == EL3 then
    BPIALL();

```


BPIALLIS, Branch Predictor Invalidate All, Inner Shareable

The BPIALLIS characteristics are:

Purpose

Invalidate all entries from branch predictors Inner Shareable.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to BPIALLIS are UNDEFINED.

In an implementation where the branch predictors are architecturally invisible, this instruction can execute as a NOP.

Attributes

BPIALLIS is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the BPIALLIS instruction

The PE ignores the value of <Rt>. Software does not have to write a value to this register before issuing this instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b0001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        BPIALLIS();
elsif PSTATE.EL == EL2 then
    BPIALLIS();
elsif PSTATE.EL == EL3 then
    BPIALLIS();

```


BPIMVA, Branch Predictor Invalidate by VA

The BPIMVA characteristics are:

Purpose

Invalidate virtual address from branch predictors.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to BPIMVA are UNDEFINED.

In an implementation where the branch predictors are architecturally invisible, this instruction can execute as a NOP.

Attributes

BPIMVA is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Virtual address to use.

Executing the BPIMVA instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b0101 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        BPIMVA(R[t]);
    elsif PSTATE.EL == EL2 then
        BPIMVA(R[t]);
    elsif PSTATE.EL == EL3 then
        BPIMVA(R[t]);

```

CCSIDR, Current Cache Size ID Register

The CCSIDR characteristics are:

Purpose

Provides information about the architecture of the currently selected cache.

When FEAT_CCIDX is implemented, this register is used in conjunction with [CCSIDR2](#).

Configuration

AArch32 System register CCSIDR bits [31:0] are architecturally mapped to AArch64 System register [CCSIDR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CCSIDR are UNDEFINED.

The implementation includes one CCSIDR for each cache that it can access. [CSSELR](#) and the Security state select which Cache Size ID Register is accessible.

Attributes

CCSIDR is a 32-bit register.

Field descriptions

When FEAT_CCIDX is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|---------------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|----------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | Associativity | | | | | | | | | | | | | | | | | | | | LineSize | | | |

Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits [31:24]

Reserved, RES0.

Associativity, bits [23:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

(Log₂(Number of bytes in cache line)) - 4. For example:

For a line length of 16 bytes: Log₂(16) = 4, LineSize entry = 0. This is the minimum line length.

For a line length of 32 bytes: Log₂(32) = 5, LineSize entry = 1.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------------|----|---|---|---|---|---|---|----------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| UNKNOWN | | | | NumSets | | | | | | | | | | | | | | | | Associativity | | | | | | | | LineSize | | | |

Note

The parameters NumSets, Associativity, and LineSize in these registers define the architecturally visible parameters that are required for the cache maintenance by Set/Way instructions. They are not guaranteed to represent the actual microarchitectural features of a design. You cannot make any inference about the actual sizes of caches based on these parameters.

Bits [31:28]

Reserved, UNKNOWN.

NumSets, bits [27:13]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Associativity, bits [12:3]

(Associativity of cache) - 1, therefore a value of 0 indicates an associativity of 1. The associativity does not have to be a power of 2.

LineSize, bits [2:0]

(Log₂(Number of bytes in cache line)) - 4. For example:
 For a line length of 16 bytes: Log₂(16) = 4, LineSize entry = 0. This is the minimum line length.
 For a line length of 32 bytes: Log₂(32) = 5, LineSize entry = 1.

Accessing CCSIDR

If [CSSELR](#).Level is programmed to a cache level that is not implemented, then on a read of the CCSIDR the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR read is treated as NOP.
- The CCSIDR read is UNDEFINED.
- The CCSIDR read returns an UNKNOWN value.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b001 | 0b0000 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return CCSIDR;
elsif PSTATE.EL == EL2 then
    return CCSIDR;
elsif PSTATE.EL == EL3 then
    return CCSIDR;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CCSIDR2, Current Cache Size ID Register 2

The CCSIDR2 characteristics are:

Purpose

Provides information about the architecture of the currently selected cache.

Configuration

AArch32 System register CCSIDR2 bits [31:0] are architecturally mapped to AArch64 System register [CCSIDR2_EL1\[31:0\]](#).

This register is present only when FEAT_CCIDX is implemented and AArch32 is supported at EL1. Otherwise, direct accesses to CCSIDR2 are UNDEFINED.

The implementation includes one CCSIDR2 for each cache that it can access. [CSSELR](#) and the Security state select which Cache Size ID Register is accessible.

Attributes

CCSIDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | NumSets | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

NumSets, bits [23:0]

(Number of sets in cache) - 1, therefore a value of 0 indicates 1 set in the cache. The number of sets does not have to be a power of 2.

Accessing CCSIDR2

If [CSSELR](#).Level is programmed to a cache level that is not implemented, then on a read of the CCSIDR2 the behavior is CONSTRAINED UNPREDICTABLE, and can be one of the following:

- The CCSIDR2 read is treated as NOP.
- The CCSIDR2 read is UNDEFINED.
- The CCSIDR2 read returns an UNKNOWN value.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b001 | 0b0000 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return CCSIDR2;
elsif PSTATE.EL == EL2 then
    return CCSIDR2;
elsif PSTATE.EL == EL3 then
    return CCSIDR2;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CFPRCTX, Control Flow Prediction Restriction by Context

The CFPRCTX characteristics are:

Purpose

Control Flow Prediction Restriction by Context applies to all Control Flow Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

Control flow predictions determined by the actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control speculative execution occurring after the instruction is complete and synchronized.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

Configuration

This instruction is present only when AArch32 is supported and FEAT_SPECRES is implemented. Otherwise, direct accesses to CFPRCTX are UNDEFINED.

Attributes

CFPRCTX is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|-------|----|------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | GVMID | | NS | EL | VMID | | | | | | | | RES0 | | | | GASID | | ASID | | | | | | | | | |

Bits [31:28]

Reserved, RES0.

GVMID, bit [27]

Execution of this instruction applies to all VMIDs or a specified VMID.

| GVMID | Meaning |
|-------|---|
| 0b0 | Applies to specified VMID for an EL0 or EL1 target execution context. |
| 0b1 | Applies to all VMIDs for an EL0 or EL1 target execution context. |

For target execution contexts other than EL0 or EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

NS, bit [26]

Security State.

| NS | Meaning |
|-----|-------------------|
| 0b0 | Secure state. |
| 0b1 | Non-secure state. |

If the instruction is executed in Non-secure state, this field has an Effective value of 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

| EL | Meaning |
|------|---------|
| 0b00 | EL0. |
| 0b01 | EL1. |
| 0b10 | EL2. |
| 0b11 | EL3. |

If the instruction is executed at an Exception level lower than the specified level, this instruction is treated as a NOP.

VMID, bits [23:16]

Only applies when bit[27] is 0 and the target execution context is either:

- EL1.
- EL0 when ([HCR_EL2.E2H==0](#) or [HCR_EL2.TGE==0](#)) or EL2 is using AArch32 state.

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0 and ([HCR_EL2.E2H==0](#) or [HCR_EL2.TGE==0](#) or [ELUsingAArch32\(EL2\)](#)), this field is treated as the current VMID.

When the instruction is executed at EL0 and ([HCR_EL2.E2H==1](#) and [HCR_EL2.TGE==1](#) and [!ELUsingAArch32\(EL2\)](#)), this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

Bits [15:9]

Reserved, RES0.

GASID, bit [8]

Execution of this instruction applies to all ASIDs or a specified ASID.

| GASID | Meaning |
|-------|--|
| 0b0 | Applies to specified ASID for an EL0 target execution context. |
| 0b1 | Applies to all ASID for an EL0 target execution context. |

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field is treated as 0.

ASID, bits [7:0]

Only applies for an EL0 target execution context and when bit[8] is 0.

Otherwise, this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

Executing the CFPRCTX instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b0011 | 0b100 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX ==
'0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        elsif ELUsingAArch32(EL1) && SCTLR.EnRCTX == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.CFPRCTX == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.EnRCTX ==
'0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch32.RestrictPrediction(R[t], RestrictType_ControlFlow);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.NV == '1' then
            AArch64.SystemAccessTrap(EL2, 0x03);
        else
            AArch32.RestrictPrediction(R[t], RestrictType_ControlFlow);
    elsif PSTATE.EL == EL2 then
        AArch32.RestrictPrediction(R[t], RestrictType_ControlFlow);
    elsif PSTATE.EL == EL3 then
        AArch32.RestrictPrediction(R[t], RestrictType_ControlFlow);

```

CLIDR, Cache Level ID Register

The CLIDR characteristics are:

Purpose

Identifies the type of cache, or caches, that are implemented at each level and can be managed using the architected cache maintenance instructions that operate by set/way, up to a maximum of seven levels. Also identifies the Level of Coherence (LoC) and Level of Unification (LoU) for the cache hierarchy.

Configuration

AArch32 System register CLIDR bits [31:0] are architecturally mapped to AArch64 System register [CLIDR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CLIDR are UNDEFINED.

Attributes

CLIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|------|-----|-------|--------|--------|--------|--------|--------|--------|--------|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ICB | LoUU | LoC | LoUIS | Ctype7 | Ctype6 | Ctype5 | Ctype4 | Ctype3 | Ctype2 | Ctype1 | | | | | | | | | | | | | | | | | | | | | |

ICB, bits [31:30]

Inner cache boundary. This field indicates the boundary for caching Inner Cacheable memory regions.

| ICB | Meaning |
|------|--|
| 0b00 | Not disclosed by this mechanism. |
| 0b01 | L1 cache is the highest Inner Cacheable level. |
| 0b10 | L2 cache is the highest Inner Cacheable level. |
| 0b11 | L3 cache is the highest Inner Cacheable level. |

LoUU, bits [29:27]

Level of Unification Uniprocessor for the cache hierarchy.

Note

When FEAT_S2FWB is implemented, the architecture requires that this field is zero so that no levels of data cache need to be cleaned in order to manage coherency with instruction fetches.

LoC, bits [26:24]

Level of Coherence for the cache hierarchy.

LoUIS, bits [23:21]

Level of Unification Inner Shareable for the cache hierarchy.

Note

When FEAT_S2FWB is implemented, the architecture requires that this field is zero so that no levels of data cache need to be cleaned in order to manage coherency with instruction fetches.

Ctype<n>, bits [3(n-1)+2:3(n-1)], for n = 7 to 1

Cache Type fields. Indicate the type of cache that is implemented and can be managed using the architected cache maintenance instructions that operate by set/way at each level, from Level 1 up to a maximum of seven levels of cache hierarchy.

| Ctype<n> | Meaning |
|----------|---------------------------------------|
| 0b000 | No cache. |
| 0b001 | Instruction cache only. |
| 0b010 | Data cache only. |
| 0b011 | Separate instruction and data caches. |
| 0b100 | Unified cache. |

All other values are reserved.

If software reads the Cache Type fields from Ctype1 upwards, once it has seen a value of 000, no caches that can be managed using the architected cache maintenance instructions that operate by set/way exist at further-out levels of the hierarchy. So, for example, if Ctype3 is the first Cache Type field with a value of 000, the values of Ctype4 to Ctype7 must be ignored.

Accessing CLIDR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b001 | 0b0000 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return CLIDR;
elsif PSTATE.EL == EL2 then
    return CLIDR;
elsif PSTATE.EL == EL3 then
    return CLIDR;

```

CNTFRQ, Counter-timer Frequency register

The CNTFRQ characteristics are:

Purpose

This register is provided so that software can discover the frequency of the system counter. It must be programmed with this value as part of system initialization. The value of the register is not interpreted by hardware.

Configuration

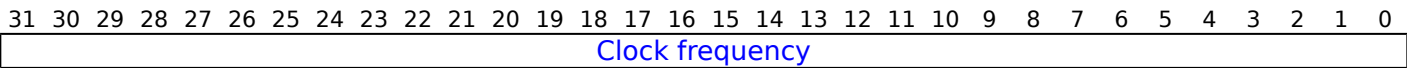
AArch32 System register CNTFRQ bits [31:0] are architecturally mapped to AArch64 System register [CNTFRQ_ELO\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTFRQ are UNDEFINED.

Attributes

CNTFRQ is a 32-bit register.

Field descriptions



Bits [31:0]

Clock frequency. Indicates the system counter clock frequency, in Hz.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTFRQ

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
    CNTKCTL_EL1.<EL0PCTEN,EL0VCTEN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PCTEN == '0' && CNTKCTL.PL0VCTEN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
            CNTHCTL_EL2.<EL0PCTEN,EL0VCTEN> == '00' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            else
                return CNTFRQ;
        elsif PSTATE.EL == EL1 then
            return CNTFRQ;
        elsif PSTATE.EL == EL2 then
            return CNTFRQ;
        elsif PSTATE.EL == EL3 then
            return CNTFRQ;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0000 | 0b000 |

```

if IsHighestEL(PSTATE.EL) then
    CNTFRQ = R[t];
else
    UNDEFINED;

```

CNTHCTL, Counter-timer Hyp Control register

The CNTHCTL characteristics are:

Purpose

Controls the generation of an event stream from the physical counter, and access from Non-secure EL1 modes to the physical counter and the Non-secure EL1 physical timer.

Configuration

AArch32 System register CNTHCTL bits [31:0] are architecturally mapped to AArch64 System register [CNTHCTL_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTHCTL are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHCTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|------|----|----|----|----|----|----|-------|---------|--------|---------|----------|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | EVNTIS | RES0 | | | | | | | EVNTI | EVNTDIR | EVNTEN | PL1PCEN | PL1PCTEN | | | | | |

Bits [31:18]

Reserved, RES0.

EVNTIS, bit [17]

When FEAT_ECV is implemented:

Controls the scale of the generation of the event stream.

| EVNTIS | Meaning |
|--------|---|
| 0b0 | The CNTHCTL.EVNTI field applies to CNTPCT[15:0] . |
| 0b1 | The CNTHCTL.EVNTI field applies to CNTPCT[23:8] . |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:8]

Reserved, RES0.

EVNTI, bits [7:4]

Selects which bit of [CNTPCT](#), as seen from EL2, is the trigger for the event stream generated from that counter when that stream is enabled.

If FEAT_ECV is implemented, and CNTHCTL.EVNTIS is 1, this field selects a trigger bit in the range 8 to 23 of [CNTPCT](#).

Otherwise, this field selects a trigger bit in the range 0 to 15 of [CNTPCT](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the [CNTPCT](#) trigger bit, as seen from EL2 and defined by EVNTI, generates an event when the event stream is enabled.

| EVNTDIR | Meaning |
|---------|---|
| 0b0 | A 0 to 1 transition of the trigger bit triggers an event. |
| 0b1 | A 1 to 0 transition of the trigger bit triggers an event. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

Enables the generation of an event stream from [CNTPCT](#) as seen from EL2.

| EVNTEN | Meaning |
|--------|----------------------------|
| 0b0 | Disables the event stream. |
| 0b1 | Enables the event stream. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PL1PCEN, bit [1]

Traps Non-secure EL0 and EL1 accesses to the physical timer registers to Hyp mode.

| PL1PCEN | Meaning |
|---------|--|
| 0b0 | Non-secure EL0 and EL1 accesses to the CNTP_CTL , CNTP_CVAL , and CNTP_TVAL are trapped to Hyp mode, unless the it is trapped by CNTKCTL .PLOPTEN. |
| 0b1 | This control does not cause any instructions to be trapped. |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PL1PCTEN, bit [0]

Traps Non-secure EL0 and EL1 accesses to the physical counter register to Hyp mode.

| PL1PCTEN | Meaning |
|----------|--|
| 0b0 | Non-secure EL0 and EL1 accesses to the CNTPCT are trapped to Hyp mode, unless it is trapped by CNTKCTL . PL0PCTEN. |
| 0b1 | This control does not cause any instructions to be trapped. |

If EL3 is implemented and EL2 is not implemented, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHCTL

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1110 | 0b0001 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHCTL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return CNTHCTL;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1110 | 0b0001 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHCTL = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        CNTHCTL = R[t];
```

CNTHP_CTL, Counter-timer Hyp Physical Timer Control register

The CNTHP_CTL characteristics are:

Purpose

Control register for the Hyp mode physical timer.

Configuration

AArch32 System register CNTHP_CTL bits [31:0] are architecturally mapped to AArch64 System register [CNTHP_CTL_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTHP_CTL are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHP_CTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|-------|----|--------|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | ISTATUS | | | IMASK | | ENABLE | | | | | | | | | | |

Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTHP_TVAL](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, when the PE resets into EL2 or EL3, this field resets to 0.

Accessing CNTHP_CTL

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHP_CTL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return CNTHP_CTL;

```

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CTL = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        CNTHP_CTL = R[t];

```

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CTL_EL2;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
    then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CNTP_CTL_NS;
    else
        return CNTP_CTL;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            return CNTP_CTL_NS;
        else
            return CNTP_CTL;
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' then
            return CNTP_CTL_S;
        else
            return CNTP_CTL_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
    == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
    && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
    then
        CNTHP_CTL_EL2 = R[t];
    else
        CNTP_CTL = R[t];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            CNTP_CTL_NS = R[t];
        else
            CNTP_CTL = R[t];
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' then
            CNTP_CTL_S = R[t];
        else
            CNTP_CTL_NS = R[t];

```

CNTHP_CVAL, Counter-timer Hyp Physical CompareValue register

The CNTHP_CVAL characteristics are:

Purpose

Holds the compare value for the Hyp mode physical timer.

Configuration

AArch32 System register CNTHP_CVAL bits [63:0] are architecturally mapped to AArch64 System register [CNTHP_CVAL_EL2\[63:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTHP_CVAL are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHP_CVAL is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | CompareValue | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | CompareValue | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

CompareValue, bits [63:0]

Holds the EL2 physical timer CompareValue.

When [CNTHP_CTL.ENABLE](#) is 1, the timer condition is met when ([CNTPCT](#) - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTHP_CTL.ISTATUS](#) is set to 1.
- If [CNTHP_CTL.IMASK](#) is 0, an interrupt is generated.

When [CNTHP_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTPCT](#) continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHP_CVAL

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| | | |
|--------|-----|------|
| coproc | CRm | opc1 |
|--------|-----|------|

| | | |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0110 |
|--------|--------|--------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTHP_CVAL;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return CNTHP_CVAL;

```

MCCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|---------------|------------|-------------|
| 0b1111 | 0b1110 | 0b0110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CVAL = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        CNTHP_CVAL = R[t2]:R[t];

```

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|---------------|------------|-------------|
| 0b1111 | 0b1110 | 0b0010 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
            AArch32.TakeHypTrapException(0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
            return CNTHPS_CVAL_EL2;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
            return CNTHP_CVAL_EL2;
        else
            return CNTP_CVAL;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
            AArch32.TakeHypTrapException(0x04);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
            return CNTP_CVAL_NS;
        else
            return CNTP_CVAL;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            return CNTP_CVAL_NS;
        else
            return CNTP_CVAL;
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' then
            return CNTP_CVAL_S;
        else
            return CNTP_CVAL_NS;

```

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0010 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
    == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
    && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = R[t2]:R[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
    then
        CNTHP_CVAL_EL2 = R[t2]:R[t];
    else
        CNTP_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = R[t2]:R[t];
    else
        CNTP_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = R[t2]:R[t];
    else
        CNTP_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CVAL_S = R[t2]:R[t];
    else
        CNTP_CVAL_NS = R[t2]:R[t];

```

CNTHP_TVAL, Counter-timer Hyp Physical Timer TimerValue register

The CNTHP_TVAL characteristics are:

Purpose

Holds the timer value for the Hyp mode physical timer.

Configuration

AArch32 System register CNTHP_TVAL bits [31:0] are architecturally mapped to AArch64 System register [CNTHP_TVAL_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTHP_TVAL are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

CNTHP_TVAL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | TimerValue | | | | | | | | | | | | | | | |

TimerValue, bits [31:0]

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If [CNTHP_CTL.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTHP_CTL.ENABLE](#) is 1, the value returned is ([CNTHP_CVAL](#) - [CNTPCT](#)).

On a write of this register, [CNTHP_CVAL](#) is set to ([CNTPCT](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTHP_CTL.ENABLE](#) is 1, the timer condition is met when ([CNTPCT](#) - [CNTHP_CVAL](#)) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTHP_CTL.ISTATUS](#) is set to 1.
- If [CNTHP_CTL.IMASK](#) is 0, an interrupt is generated.

When [CNTHP_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTPCT](#) continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHP_TVAL

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    if CNTHP_CTL.ENABLE == '0' then
        return bits(32) UNKNOWN;
    else
        return (CNTHP_CVAL - PhysicalCountInt())<31:0>;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        if CNTHP_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTHP_CVAL - PhysicalCountInt())<31:0>;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTHP_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        CNTHP_CVAL = SignExtend(R[t],64) + PhysicalCountInt();

```

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
== '1' && CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
        if CNTP_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>;
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '1' then
            if CNTP_CTL_NS.ENABLE == '0' then
                return bits(32) UNKNOWN;
            else
                return (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
        else
            if CNTP_CTL_S.ENABLE == '0' then
                return bits(32) UNKNOWN;
            else
                return (CNTP_CVAL_S - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL - PhysicalCountInt())<31:0>;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
== '1' && CNTHCTL_EL2.ECV == '1' then

```

```

    if CNTP_CTL.ENABLE == '0' then
        return bits(32) UNKNOWN;
    else
        return (CNTP_CVAL - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>;
elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if CNTP_CTL_NS.ENABLE == '0' then
        return bits(32) UNKNOWN;
    else
        return (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
else
    if CNTP_CTL.ENABLE == '0' then
        return bits(32) UNKNOWN;
    else
        return (CNTP_CVAL - PhysicalCountInt())<31:0>;
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        if CNTP_CTL_NS.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL - PhysicalCountInt())<31:0>;
elseif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        if CNTP_CTL_S.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL_S - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL_NS.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend(R[t],64) + PhysicalCountInt();
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHP_CVAL_EL2 = SignExtend(R[t],64) + PhysicalCountInt();
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
== '1' && CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt() - CNTPOFF_EL2;
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '1' then
            CNTP_CVAL_NS = SignExtend(R[t],64) + PhysicalCountInt();
        else
            CNTP_CVAL_S = SignExtend(R[t],64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
== '1' && CNTHCTL_EL2.ECV == '1' then
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt() - CNTPOFF_EL2;
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = SignExtend(R[t],64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = SignExtend(R[t],64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
elseif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CVAL_S = SignExtend(R[t],64) + PhysicalCountInt();
    else
        CNTP_CVAL_NS = SignExtend(R[t],64) + PhysicalCountInt();

```


CNTHPS_CTL, Counter-timer Secure Physical Timer Control Register (EL2)

The CNTHPS_CTL characteristics are:

Purpose

Provides AArch32 access from EL0 to the Secure EL2 physical timer.

Configuration

AArch32 System register CNTHPS_CTL bits [31:0] are architecturally mapped to AArch64 System register [CNTHPS_CTL_EL2\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHPS_CTL are UNDEFINED.

Attributes

CNTHPS_CTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|-------|---|--------|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | ISTATUS | | | IMASK | | ENABLE | | | | | | | | |

Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the CNTHPS_CTL.ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the CNTHPS_CTL.ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTHPS_TVAL_EL2](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_CTL

This register is accessed using the encoding for [CNTP_CTL](#).

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
            return CNTHPS_CTL_EL2;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
            return CNTHP_CTL_EL2;
        else
            return CNTP_CTL;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
            AArch32.TakeHypTrapException(0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
            return CNTP_CTL_NS;
        else
            return CNTP_CTL;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            return CNTP_CTL_NS;
        else
            return CNTP_CTL;
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' then
            return CNTP_CTL_S;
        else
            return CNTP_CTL_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
    == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
    && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = R[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
    then
        CNTHP_CTL_EL2 = R[t];
    else
        CNTP_CTL = R[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
elseif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CTL_S = R[t];
    else
        CNTP_CTL_NS = R[t];

```

CNTHPS_CVAL, Counter-timer Secure Physical Timer CompareValue Register (EL2)

The CNTHPS_CVAL characteristics are:

Purpose

Provides AArch32 access from EL0 to the compare value for the Secure EL2 physical timer.

Configuration

AArch32 System register CNTHPS_CVAL bits [63:0] are architecturally mapped to AArch64 System register [CNTHPS_CVAL_EL2\[63:0\]](#).

This register is present only when AArch32 is supported and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHPS_CVAL are UNDEFINED.

Attributes

CNTHPS_CVAL is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| | | | | | | | | | | | | | | | | CompareValue | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | CompareValue | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

CompareValue, bits [63:0]

Holds the EL2 physical timer CompareValue.

When `CNTHPS_CTL_EL2.ENABLE` is 1, the timer condition is met when `(CNTPCT_EL0 - CompareValue)` is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTHPS_CTL_EL2](#).ISTATUS is set to 1.
- If [CNTHPS_CTL_EL2](#).IMASK is 0, an interrupt is generated.

When [CNTHPS_CTL_EL2.ENABLE](#) is 0, the timer condition is not met, but [CNTPTCT_EL0](#) continues to count

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS CVAL

This register is accessed using the encoding for [CNTF_CVAL](#).

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0010 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
            AArch32.TakeHypTrapException(0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
            return CNTHPS_CVAL_EL2;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
            return CNTHP_CVAL_EL2;
        else
            return CNTP_CVAL;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
            AArch32.TakeHypTrapException(0x04);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
            return CNTP_CVAL_NS;
        else
            return CNTP_CVAL;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            return CNTP_CVAL_NS;
        else
            return CNTP_CVAL;
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' then
            return CNTP_CVAL_S;
        else
            return CNTP_CVAL_NS;

```

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0010 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
    == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
    && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = R[t2]:R[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
    then
        CNTHP_CVAL_EL2 = R[t2]:R[t];
    else
        CNTP_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = R[t2]:R[t];
    else
        CNTP_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = R[t2]:R[t];
    else
        CNTP_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CVAL_S = R[t2]:R[t];
    else
        CNTP_CVAL_NS = R[t2]:R[t];

```

CNTHPS_TVAL, Counter-timer Secure Physical Timer TimerValue Register (EL2)

The CNTHPS_TVAL characteristics are:

Purpose

Provides AArch32 access from EL0 to the timer value for the Secure EL2 physical timer.

Configuration

AArch32 System register CNTHPS_TVAL bits [31:0] are architecturally mapped to AArch64 System register [CNTHPS_TVAL_EL2\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHPS_TVAL are UNDEFINED.

Attributes

CNTHPS_TVAL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TimerValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

TimerValue, bits [31:0]

The TimerValue view of the EL2 physical timer.

On a read of this register:

- If [CNTHPS_CTL_EL2.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTHPS_CTL_EL2.ENABLE](#) is 1, the value returned is ([CNTHPS_CVAL_EL2](#) - [CNTPCT_EL0](#)).

On a write of this register, [CNTHPS_CVAL_EL2](#) is set to ([CNTPCT_EL0](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTHPS_CTL_EL2.ENABLE](#) is 1, the timer condition is met when ([CNTPCT_EL0](#) - [CNTHPS_CVAL_EL2](#)) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTHPS_CTL_EL2.ISTATUS](#) is set to 1.
- If [CNTHPS_CTL_EL2.IMASK](#) is 0, an interrupt is generated.

When [CNTHPS_CTL_EL2.ENABLE](#) is 0, the timer condition is not met, but [CNTPCT_EL0](#) continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHPS_TVAL

This register is accessed using the encoding for [CNTP_TVAL](#).

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
    == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
    && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
    then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elsif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEN
    == '1' && CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
        if CNTP_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>;
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '1' then
            if CNTP_CTL_NS.ENABLE == '0' then
                return bits(32) UNKNOWN;
            else
                return (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
        else
            if CNTP_CTL_S.ENABLE == '0' then
                return bits(32) UNKNOWN;
            else
                return (CNTP_CVAL_S - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL - PhysicalCountInt())<31:0>;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEN
    == '1' && CNTHCTL_EL2.ECV == '1' then

```

```

    if CNTP_CTL.ENABLE == '0' then
        return bits(32) UNKNOWN;
    else
        return (CNTP_CVAL - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>;
elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if CNTP_CTL_NS.ENABLE == '0' then
        return bits(32) UNKNOWN;
    else
        return (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
else
    if CNTP_CTL.ENABLE == '0' then
        return bits(32) UNKNOWN;
    else
        return (CNTP_CVAL - PhysicalCountInt())<31:0>;
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        if CNTP_CTL_NS.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL - PhysicalCountInt())<31:0>;
elseif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        if CNTP_CTL_S.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL_S - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL_NS.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend(R[t],64) + PhysicalCountInt();
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHP_CVAL_EL2 = SignExtend(R[t],64) + PhysicalCountInt();
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
== '1' && CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt() - CNTPOFF_EL2;
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '1' then
            CNTP_CVAL_NS = SignExtend(R[t],64) + PhysicalCountInt();
        else
            CNTP_CVAL_S = SignExtend(R[t],64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
== '1' && CNTHCTL_EL2.ECV == '1' then
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt() - CNTPOFF_EL2;
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = SignExtend(R[t],64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = SignExtend(R[t],64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
elseif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CVAL_S = SignExtend(R[t],64) + PhysicalCountInt();
    else
        CNTP_CVAL_NS = SignExtend(R[t],64) + PhysicalCountInt();

```


CNTHV_CTL, Counter-timer Virtual Timer Control register (EL2)

The CNTHV_CTL characteristics are:

Purpose

Provides AArch32 access to the control register for the EL2 virtual timer.

Configuration

AArch32 System register CNTHV_CTL bits [31:0] are architecturally mapped to AArch64 System register [CNTHV_CTL_EL2\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_VHE is implemented. Otherwise, direct accesses to CNTHV_CTL are UNDEFINED.

Attributes

CNTHV_CTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|-------|---|--------|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | ISTATUS | | | IMASK | | ENABLE | | | | | | | | |

Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTHV_TVAL](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHV_CTL

This register is accessed using the encoding for [CNTV_CTL](#).

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CTL_EL2;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        return CNTV_CTL;
elseif PSTATE.EL == EL2 then
    return CNTV_CTL;
elseif PSTATE.EL == EL3 then
    return CNTV_CTL;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = R[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CTL_EL2 = R[t];
    else
        CNTV_CTL = R[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        CNTV_CTL = R[t];
elseif PSTATE.EL == EL2 then
    CNTV_CTL = R[t];
elseif PSTATE.EL == EL3 then
    CNTV_CTL = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHV_CVAL, Counter-timer Virtual Timer CompareValue register (EL2)

The CNTHV_CVAL characteristics are:

Purpose

Provides AArch32 access to the compare value for the EL2 virtual timer.

Configuration

AArch32 System register CNTHV_CVAL bits [63:0] are architecturally mapped to AArch64 System register [CNTHV_CVAL_EL2\[63:0\]](#).

This register is present only when FEAT_VHE is implemented. Otherwise, direct accesses to CNTHV_CVAL are UNDEFINED.

Attributes

CNTHV_CVAL is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

CompareValue, bits [63:0]

Holds the EL2 virtual timer CompareValue.

When [CNTHV_CTL.ENABLE](#) is 1, the timer condition is met when ([CNTVCT](#) - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTHV_CTL.ISTATUS](#) is set to 1.
- If [CNTHV_CTL.IMASK](#) is 0, an interrupt is generated.

When [CNTHV_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTVCT](#) continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

Accessing CNTHV_CVAL

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0011 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CVAL_EL2;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        return CNTV_CVAL;
elseif PSTATE.EL == EL2 then
    return CNTV_CVAL;
elseif PSTATE.EL == EL3 then
    return CNTV_CVAL;

```

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0011 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = R[t2]:R[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CVAL_EL2 = R[t2]:R[t];
    else
        CNTV_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        CNTV_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL2 then
    CNTV_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL3 then
    CNTV_CVAL = R[t2]:R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHV_TVAL, Counter-timer Virtual Timer TimerValue register (EL2)

The CNTHV_TVAL characteristics are:

Purpose

Provides AArch32 access to the timer value for the EL2 virtual timer.

Configuration

AArch32 System register CNTHV_TVAL bits [31:0] are architecturally mapped to AArch64 System register [CNTHV_TVAL_EL2\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_VHE is implemented. Otherwise, direct accesses to CNTHV_TVAL are UNDEFINED.

Attributes

CNTHV_TVAL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TimerValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

TimerValue, bits [31:0]

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If [CNTHV_CTL.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTHV_CTL.ENABLE](#) is 1, the value returned is ([CNTHV_CVAL](#) - [CNTVCT](#)).

On a write of this register, [CNTHV_CVAL](#) is set to ([CNTVCT](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTHV_CTL.ENABLE](#) is 1, the timer condition is met when ([CNTVCT](#) - [CNTHV_CVAL](#)) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTHV_CTL.ISTATUS](#) is set to 1.
- If [CNTHV_CTL.IMASK](#) is 0, an interrupt is generated.

When [CNTHV_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTVCT](#) continues to count, so the TimerValue view appears to continue to count down.

Accessing CNTHV_TVAL

This register is accessed using the encoding for [CNTV_TVAL](#).

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b000 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>;
    else
        if CNTV_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        elsif HaveEL(EL2) && !ELUsingAArch32(EL2) then
            return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>;
        elsif HaveEL(EL2) && ELUsingAArch32(EL2) then
            return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
        else
            return (CNTV_CVAL - PhysicalCountInt())<31:0>;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            if CNTV_CTL.ENABLE == '0' then
                return bits(32) UNKNOWN;
            elsif HaveEL(EL2) && !ELUsingAArch32(EL2) then
                return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>;
            elsif HaveEL(EL2) && ELUsingAArch32(EL2) then
                return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
            else
                return (CNTV_CVAL - PhysicalCountInt())<31:0>;
    elsif PSTATE.EL == EL2 then
        if CNTV_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
    elsif PSTATE.EL == EL3 then
        if CNTV_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        elsif HaveEL(EL2) then
            return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
        else
            return (CNTV_CVAL - PhysicalCountInt())<31:0>;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend(R[t],64) + PhysicalCountInt();
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CVAL_EL2 = SignExtend(R[t],64) + PhysicalCountInt();
    else
        if HaveEL(EL2) && !ELUsingAArch32(EL2) then
            CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt() - CNTVOFF_EL2;
        elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
            CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt() - CNTVOFF;
        else
            CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            if HaveEL(EL2) && !ELUsingAArch32(EL2) then
                CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt() - CNTVOFF_EL2;
            elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
                CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt() - CNTVOFF;
            else
                CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
    elseif PSTATE.EL == EL2 then
        CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt() - CNTVOFF;
    elseif PSTATE.EL == EL3 then
        if HaveEL(EL2) then
            CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt() - CNTVOFF;
        else
            CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt();

```

CNTHVS_CTL, Counter-timer Secure Virtual Timer Control Register (EL2)

The CNTHVS_CTL characteristics are:

Purpose

Provides AArch32 access from EL0 to the Secure EL2 virtual timer.

Configuration

AArch32 System register CNTHVS_CTL bits [31:0] are architecturally mapped to AArch64 System register [CNTHVS_CTL_EL2\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHVS_CTL are UNDEFINED.

Attributes

CNTHVS_CTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|-------|---|--------|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | ISTATUS | | | IMASK | | ENABLE | | | | | | | | |

Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTHVS_TVAL](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTHVS_CTL

This register is accessed using the encoding for [CNTV_CTL](#).

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CTL_EL2;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        return CNTV_CTL;
elseif PSTATE.EL == EL2 then
    return CNTV_CTL;
elseif PSTATE.EL == EL3 then
    return CNTV_CTL;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = R[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CTL_EL2 = R[t];
    else
        CNTV_CTL = R[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        CNTV_CTL = R[t];
elseif PSTATE.EL == EL2 then
    CNTV_CTL = R[t];
elseif PSTATE.EL == EL3 then
    CNTV_CTL = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHVS_CVAL, Counter-timer Secure Virtual Timer CompareValue Register (EL2)

The CNTHVS_CVAL characteristics are:

Purpose

Provides AArch32 access to the compare value for the Secure EL2 virtual timer.

Configuration

AArch32 System register CNTHVS_CVAL bits [63:0] are architecturally mapped to AArch64 System register [CNTHVS_CVAL_EL2\[63:0\]](#).

This register is present only when AArch32 is supported and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHVS_CVAL are UNDEFINED.

Attributes

CNTHVS_CVAL is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | |

CompareValue, bits [63:0]

Holds the EL2 virtual timer CompareValue.

When [CNTHVS_CTL.ENABLE](#) is 1, the timer condition is met when ([CNTVCT](#) - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTHVS_CTL.ISTATUS](#) is set to 1.
- If [CNTHVS_CTL.IMASK](#) is 0, an interrupt is generated.

When [CNTHVS_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTVCT](#) continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

Accessing CNTHVS_CVAL

This register is accessed using the encoding for [CNTV_CVAL](#).

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0011 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elseif ELUsingAArch32(EL1) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CVAL_EL2;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        return CNTV_CVAL;
elseif PSTATE.EL == EL2 then
    return CNTV_CVAL;
elseif PSTATE.EL == EL3 then
    return CNTV_CVAL;

```

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0011 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elseif ELUsingAArch32(EL1) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = R[t2]:R[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CVAL_EL2 = R[t2]:R[t];
    else
        CNTV_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        CNTV_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL2 then
    CNTV_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL3 then
    CNTV_CVAL = R[t2]:R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTHVS_TVAL, Counter-timer Secure Virtual Timer TimerValue Register (EL2)

The CNTHVS_TVAL characteristics are:

Purpose

Provides AArch32 access to the timer value for the Secure EL2 virtual timer.

Configuration

AArch32 System register CNTHVS_TVAL bits [31:0] are architecturally mapped to AArch64 System register [CNTHVS_TVAL_EL2\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_SEL2 is implemented. Otherwise, direct accesses to CNTHVS_TVAL are UNDEFINED.

Attributes

CNTHVS_TVAL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TimerValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

TimerValue, bits [31:0]

The TimerValue view of the EL2 virtual timer.

On a read of this register:

- If [CNTHVS_CTL.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTHVS_CTL.ENABLE](#) is 1, the value returned is ([CNTHVS_CVAL](#) - [CNTVCT](#)).

On a write of this register, [CNTHVS_CVAL](#) is set to ([CNTVCT](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTHVS_CTL.ENABLE](#) is 1, the timer condition is met when ([CNTVCT](#) - [CNTHVS_CVAL](#)) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTHVS_CTL.ISTATUS](#) is set to 1.
- If [CNTHVS_CTL.IMASK](#) is 0, an interrupt is generated.

When [CNTHVS_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTVCT](#) continues to count, so the TimerValue view appears to continue to count down.

Accessing CNTHVS_TVAL

This register is accessed using the encoding for [CNTV_TVAL](#).

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b000 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>;
    else
        if CNTV_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        elseif HaveEL(EL2) && !ELUsingAArch32(EL2) then
            return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>;
        elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
            return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
        else
            return (CNTV_CVAL - PhysicalCountInt())<31:0>;
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            if CNTV_CTL.ENABLE == '0' then
                return bits(32) UNKNOWN;
            elseif HaveEL(EL2) && !ELUsingAArch32(EL2) then
                return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>;
            elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
                return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
            else
                return (CNTV_CVAL - PhysicalCountInt())<31:0>;
    elseif PSTATE.EL == EL2 then
        if CNTV_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
    elseif PSTATE.EL == EL3 then
        if CNTV_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        elseif HaveEL(EL2) then
            return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
        else
            return (CNTV_CVAL - PhysicalCountInt())<31:0>;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend(R[t],64) + PhysicalCountInt();
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CVAL_EL2 = SignExtend(R[t],64) + PhysicalCountInt();
    else
        if HaveEL(EL2) && !ELUsingAArch32(EL2) then
            CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt() - CNTVOFF_EL2 ;
        elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
            CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt() - CNTVOFF;
        else
            CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            if HaveEL(EL2) && !ELUsingAArch32(EL2) then
                CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt() - CNTVOFF_EL2 ;
            elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
                CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt() - CNTVOFF;
            else
                CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
    elseif PSTATE.EL == EL2 then
        CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt() - CNTVOFF;
    elseif PSTATE.EL == EL3 then
        if HaveEL(EL2) then
            CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt() - CNTVOFF;
        else
            CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt();

```

CNTKCTL, Counter-timer Kernel Control register

The CNTKCTL characteristics are:

Purpose

Controls the generation of an event stream from the virtual counter, and access from EL0 modes to the physical counter, virtual counter, EL1 physical timers, and the virtual timer.

Configuration

AArch32 System register CNTKCTL bits [31:0] are architecturally mapped to AArch64 System register [CNTKCTL_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTKCTL are UNDEFINED.

Attributes

CNTKCTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|------|----|----|----|----|----|---------|---------|-------|---------|--------|----------|----------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | EVNTIS | RES0 | | | | | | PLOPTEN | PLOVTEN | EVNTI | EVNTDIR | EVNTEN | PLOVCTEN | PLOPCTEN | | | | |

Bits [31:18]

Reserved, RES0.

EVENTIS, bit [17]

When FEAT_ECV is implemented:

Controls the scale of the generation of the event stream.

| EVENTIS | Meaning |
|---------|---|
| 0b0 | The CNTKCTL.EVNTI field applies to CNTVCT[15:0] . |
| 0b1 | The CNTKCTL.EVNTI field applies to CNTVCT[23:8] . |

This control applies regardless of the value of the [CNTHCTL_EL2](#).ECV bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [16:10]

Reserved, RES0.

PLOPTEN, bit [9]

Traps PL0 accesses to the physical timer registers to Undefined mode.

| PLOPTEN | Meaning |
|----------------|---|
| 0b0 | PL0 accesses to the CNTP_CTL , CNTP_CVAL , and CNTP_TVAL registers are trapped to Undefined mode. |
| 0b1 | This control does not cause any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PLOVTEN, bit [8]

Traps PL0 accesses to the virtual timer registers to Undefined mode.

| PLOVTEN | Meaning |
|----------------|---|
| 0b0 | PL0 accesses to the CNTV_CTL , CNTV_CVAL , and CNTV_TVAL registers are trapped to Undefined mode. |
| 0b1 | This control does not cause any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTI, bits [7:4]

Selects which bit of [CNTVCT](#), as seen from EL1, is the trigger for the event stream generated from that counter when that stream is enabled.

If FEAT_ECV is implemented, and CNTKCTL.EVENTIS is 1, this field selects a trigger bit in the range 8 to 23 of [CNTVCT](#).

Otherwise, this field selects a trigger bit in the range 0 to 15 of [CNTVCT](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTDIR, bit [3]

Controls which transition of the [CNTVCT](#) trigger bit, as seen from EL1 and defined by EVNTI, generates an event when the event stream is enabled.

| EVNTDIR | Meaning |
|----------------|---|
| 0b0 | A 0 to 1 transition of the trigger bit triggers an event. |
| 0b1 | A 1 to 0 transition of the trigger bit triggers an event. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EVNTEN, bit [2]

Enables the generation of an event stream from [CNTVCT](#) as seen from EL1.

| EVNTEN | Meaning |
|---------------|----------------------------|
| 0b0 | Disables the event stream. |
| 0b1 | Enables the event stream. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PLOVCTEN, bit [1]

Traps PL0 accesses to the frequency register and virtual counter register to Undefined mode.

| PLOVCTEN | Meaning |
|-----------------|---|
| 0b0 | PL0 accesses to the CNTVCT are trapped to Undefined mode. PL0 accesses to the CNTFRQ register are trapped to Undefined mode, if CNTKCTL .PLOPCTEN is also 0. |
| 0b1 | This control does not cause any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

PLOPCTEN, bit [0]

Traps PL0 accesses to the frequency register and physical counter register to Undefined mode.

| PLOPCTEN | Meaning |
|-----------------|---|
| 0b0 | PL0 accesses to the CNTPCT are trapped to Undefined mode. PL0 accesses to the CNTFRQ register are trapped to Undefined mode, if CNTKCTL .PLOVCTEN is also 0. |
| 0b1 | This control does not cause any instructions to be trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTKCTL

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1110 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    return CNTKCTL;
elsif PSTATE.EL == EL2 then
    return CNTKCTL;
elsif PSTATE.EL == EL3 then
    return CNTKCTL;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1110 | 0b0001 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    CNTKCTL = R[t];
elsif PSTATE.EL == EL2 then
    CNTKCTL = R[t];
elsif PSTATE.EL == EL3 then
    CNTKCTL = R[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_CTL, Counter-timer Physical Timer Control register

The CNTP_CTL characteristics are:

Purpose

Control register for the EL1 physical timer.

Configuration

AArch32 System register CNTP_CTL bits [31:0] are architecturally mapped to AArch64 System register [CNTP_CTL_EL0\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTP_CTL are UNDEFINED.

Attributes

CNTP_CTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|-------|----|--------|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | ISTATUS | | | IMASK | | ENABLE | | | | | | | | | | |

Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTP_TVAL](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing CNTP_CTL

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CTL_EL2;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHP_CTL_EL2;
    else
        return CNTP_CTL;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CNTP_CTL_NS;
    else
        return CNTP_CTL;
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CNTP_CTL_NS;
    else
        return CNTP_CTL;
elseif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return CNTP_CTL_S;
    else
        return CNTP_CTL_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
    == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
    && IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CTL_EL2 = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
    then
        CNTHP_CTL_EL2 = R[t];
    else
        CNTP_CTL = R[t];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CTL_NS = R[t];
    else
        CNTP_CTL = R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            CNTP_CTL_NS = R[t];
        else
            CNTP_CTL = R[t];
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' then
            CNTP_CTL_S = R[t];
        else
            CNTP_CTL_NS = R[t];

```

CNTP_CVAL, Counter-timer Physical Timer CompareValue register

The CNTP_CVAL characteristics are:

Purpose

Holds the compare value for the EL1 physical timer.

Configuration

AArch32 System register CNTP_CVAL bits [63:0] are architecturally mapped to AArch64 System register [CNTP_CVAL_EL0\[63:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTP_CVAL are UNDEFINED.

Attributes

CNTP_CVAL is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

CompareValue, bits [63:0]

Holds the EL1 physical timer CompareValue.

When [CNTP_CTL.ENABLE](#) is 1, the timer condition is met when ([CNTPCT](#) - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTP_CTL.ISTATUS](#) is set to 1.
- If [CNTP_CTL.IMASK](#) is 0, an interrupt is generated.

When [CNTP_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTPCT](#) continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_CVAL

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0010 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
        return CNTHPS_CVAL_EL2;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1' then
        return CNTHP_CVAL_EL2;
    else
        return CNTP_CVAL;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x04);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CNTP_CVAL_NS;
    else
        return CNTP_CVAL;
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CNTP_CVAL_NS;
    else
        return CNTP_CVAL;
elseif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return CNTP_CVAL_S;
    else
        return CNTP_CVAL_NS;

```

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0010 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
            AArch32.TakeHypTrapException(0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0' && IsFeatureImplemented(FEAT_SEL2) then
            CNTHPS_CVAL_EL2 = R[t2]:R[t];
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
            CNTHP_CVAL_EL2 = R[t2]:R[t];
        else
            CNTP_CVAL = R[t2]:R[t];
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
            AArch32.TakeHypTrapException(0x04);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
            CNTP_CVAL_NS = R[t2]:R[t];
        else
            CNTP_CVAL = R[t2]:R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            CNTP_CVAL_NS = R[t2]:R[t];
        else
            CNTP_CVAL = R[t2]:R[t];
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' then
            CNTP_CVAL_S = R[t2]:R[t];
        else
            CNTP_CVAL_NS = R[t2]:R[t];

```

CNTP_TVAL, Counter-timer Physical Timer TimerValue register

The CNTP_TVAL characteristics are:

Purpose

Holds the timer value for the EL1 physical timer.

Configuration

AArch32 System register CNTP_TVAL bits [31:0] are architecturally mapped to AArch64 System register [CNTP_TVAL_EL0\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTP_TVAL are UNDEFINED.

Attributes

CNTP_TVAL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TimerValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

TimerValue, bits [31:0]

The TimerValue view of the EL1 physical timer.

On a read of this register:

- If [CNTP_CTL.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTP_CTL.ENABLE](#) is 1, the value returned is ([CNTP_CVAL](#) - [CNTPTCT](#)).

On a write of this register, [CNTP_CVAL](#) is set to ([CNTPTCT](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTP_CTL.ENABLE](#) is 1, the timer condition is met when ([CNTPTCT](#) - [CNTP_CVAL](#)) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTP_CTL.ISTATUS](#) is set to 1.
- If [CNTP_CTL.IMASK](#) is 0, an interrupt is generated.

When [CNTP_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTPTCT](#) continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_TVAL

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| | | | | |
|--------|------|-----|-----|------|
| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b000 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
    == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
    == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
    && IsFeatureImplemented(FEAT_SEL2) then
        if CNTHPS_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTHPS_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
    then
        if CNTHP_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTHP_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
    == '1' && CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
        if CNTP_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>;
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '1' then
            if CNTP_CTL_NS.ENABLE == '0' then
                return bits(32) UNKNOWN;
            else
                return (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
        else
            if CNTP_CTL_S.ENABLE == '0' then
                return bits(32) UNKNOWN;
            else
                return (CNTP_CVAL_S - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL - PhysicalCountInt())<31:0>;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
    == '1' && CNTHCTL_EL2.ECV == '1' then

```

```

    if CNTP_CTL.ENABLE == '0' then
        return bits(32) UNKNOWN;
    else
        return (CNTP_CVAL - (PhysicalCountInt() - CNTPOFF_EL2))<31:0>;
elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
    if CNTP_CTL_NS.ENABLE == '0' then
        return bits(32) UNKNOWN;
    else
        return (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
else
    if CNTP_CTL.ENABLE == '0' then
        return bits(32) UNKNOWN;
    else
        return (CNTP_CVAL - PhysicalCountInt())<31:0>;
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        if CNTP_CTL_NS.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL - PhysicalCountInt())<31:0>;
elseif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        if CNTP_CTL_S.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL_S - PhysicalCountInt())<31:0>;
    else
        if CNTP_CTL_NS.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTP_CVAL_NS - PhysicalCountInt())<31:0>;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0PTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' && CNTHCTL_EL2.EL1PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0PTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        CNTHPS_CVAL_EL2 = SignExtend(R[t],64) + PhysicalCountInt();
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHP_CVAL_EL2 = SignExtend(R[t],64) + PhysicalCountInt();
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
== '1' && CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt() - CNTPOFF_EL2;
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if SCR.NS == '1' then
            CNTP_CVAL_NS = SignExtend(R[t],64) + PhysicalCountInt();
        else
            CNTP_CVAL_S = SignExtend(R[t],64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CNTHCTL_EL2.EL1PTEN == '0'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCEN == '0' then
        AArch32.TakeHypTrapException(0x03);
    elseif IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
== '1' && CNTHCTL_EL2.ECV == '1' then
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt() - CNTPOFF_EL2;
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = SignExtend(R[t],64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CNTP_CVAL_NS = SignExtend(R[t],64) + PhysicalCountInt();
    else
        CNTP_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
elseif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CNTP_CVAL_S = SignExtend(R[t],64) + PhysicalCountInt();
    else
        CNTP_CVAL_NS = SignExtend(R[t],64) + PhysicalCountInt();

```


CNPCT, Counter-timer Physical Count register

The CNTPCT characteristics are:

Purpose

Holds the 64-bit physical count value.

Configuration

AArch32 System register CNTPCT bits [63:0] are architecturally mapped to AArch64 System register [CNPCT_EL0\[63:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTPCT are UNDEFINED.

All reads to the CNTPCT occur in program order relative to reads to [CNPCTSS](#) or CNTPCT.

Attributes

CNPCT is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Physical count value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Physical count value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPCT

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0000 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
    CNTKCTL_EL1.EL0PCTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        elsif ELUsingAArch32(EL1) && CNTKCTL.PL0PCTEN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCTEN ==
            '0' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' &&
            CNTHCTL_EL2.EL1PCTEN == '0' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
            CNTHCTL_EL2.EL0PCTEN == '0' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCTEN == '0' then
                AArch32.TakeHypTrapException(0x04);
            else
                if IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
                == '1' && CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
                    return PhysicalCountInt() - CNTPOFF_EL2;
                else
                    return PhysicalCountInt();
            elsif PSTATE.EL == EL1 then
                if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
                    AArch64.AArch32SystemAccessTrap(EL2, 0x04);
                elsif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCTEN == '0' then
                    AArch32.TakeHypTrapException(0x04);
                else
                    if IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
                    == '1' && CNTHCTL_EL2.ECV == '1' then
                        return PhysicalCountInt() - CNTPOFF_EL2;
                    else
                        return PhysicalCountInt();
            elsif PSTATE.EL == EL2 then
                return PhysicalCountInt();
            elsif PSTATE.EL == EL3 then
                return PhysicalCountInt();

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTPCTSS, Counter-timer Self-Synchronized Physical Count register

The CNTPCTSS characteristics are:

Purpose

Holds the 64-bit physical count value.

Configuration

AArch32 System register CNTPCTSS bits [63:0] are architecturally mapped to AArch64 System register [CNTPCTSS_ELO\[63:0\]](#).

This register is present only when AArch32 is supported and FEAT_ECV is implemented. Otherwise, direct accesses to CNTPCTSS are UNDEFINED.

All reads to the CNTPCTSS occur in program order relative to reads to [CNTPCT](#) or CNTPCTSS.

This register is a self-synchronised view of the [CNTPCT](#) counter, and cannot be read speculatively.

Attributes

CNTPCTSS is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Self-Synchronized Physical count value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Self-Synchronized Physical count value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

- Self-Synchronized Physical count value.
- The reset behavior of this field is:
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPCTSS

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b1000 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
    CNTKCTL_EL1.EL0PCTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
        elseif ELUsingAArch32(EL1) && CNTKCTL.PL0PCTEN == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '0' && CNTHCTL_EL2.EL1PCTEN ==
        '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '10' &&
        CNTHCTL_EL2.EL1PCTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
        CNTHCTL_EL2.EL0PCTEN == '0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCTEN == '0' then
            AArch32.TakeHypTrapException(0x04);
        else
            if IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
            == '1' && CNTHCTL_EL2.ECV == '1' && HCR_EL2.<E2H,TGE> != '11' then
                return PhysicalCountInt() - CNTPOFF_EL2;
            else
                return PhysicalCountInt();
        elseif PSTATE.EL == EL1 then
            if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1PCTEN == '0' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x04);
            elseif EL2Enabled() && ELUsingAArch32(EL2) && CNTHCTL.PL1PCTEN == '0' then
                AArch32.TakeHypTrapException(0x04);
            else
                if IsFeatureImplemented(FEAT_ECV) && EL2Enabled() && !ELUsingAArch32(EL2) && SCR_EL3.ECVEn
                == '1' && CNTHCTL_EL2.ECV == '1' then
                    return PhysicalCountInt() - CNTPOFF_EL2;
                else
                    return PhysicalCountInt();
            elseif PSTATE.EL == EL2 then
                return PhysicalCountInt();
            elseif PSTATE.EL == EL3 then
                return PhysicalCountInt();

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_CTL, Counter-timer Virtual Timer Control register

The CNTV_CTL characteristics are:

Purpose

Control register for the virtual timer.

Configuration

AArch32 System register CNTV_CTL bits [31:0] are architecturally mapped to AArch64 System register [CNTV_CTL_EL0\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTV_CTL are UNDEFINED.

Attributes

CNTV_CTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|-------|----|--------|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | ISTATUS | | | IMASK | | ENABLE | | | | | | | | | | |

Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTV_TVAL](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing CNTV_CTL

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CTL_EL2;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHV_CTL_EL2;
    else
        return CNTV_CTL;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        return CNTV_CTL;
elseif PSTATE.EL == EL2 then
    return CNTV_CTL;
elseif PSTATE.EL == EL3 then
    return CNTV_CTL;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CTL_EL2 = R[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CTL_EL2 = R[t];
    else
        CNTV_CTL = R[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        CNTV_CTL = R[t];
elseif PSTATE.EL == EL2 then
    CNTV_CTL = R[t];
elseif PSTATE.EL == EL3 then
    CNTV_CTL = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_CVAL, Counter-timer Virtual Timer CompareValue register

The CNTV_CVAL characteristics are:

Purpose

Holds the compare value for the virtual timer.

Configuration

AArch32 System register CNTV_CVAL bits [63:0] are architecturally mapped to AArch64 System register [CNTV_CVAL_EL0\[63:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTV_CVAL are UNDEFINED.

Attributes

CNTV_CVAL is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| | | | | | | | | | | | | | | | | | CompareValue | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | CompareValue | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

CompareValue, bits [63:0]

Holds the EL1 virtual timer CompareValue.

When `CNTV_CTL.ENABLE` is 1, the timer condition is met when `(CNTVCT - CompareValue)` is greater than or equal to zero. This means that `CompareValue` acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTV_CTL](#).ISTATUS is set to 1.
- If [CNTV_CTL](#).IMASK is 0, an interrupt is generated.

When [CNTV_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTVCT](#) continues to count.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_CVAL

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|---------------|------------|-------------|
| 0b1111 | 0b1110 | 0b0011 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        return CNTHVS_CVAL_EL2;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        return CNTHV_CVAL_EL2;
    else
        return CNTV_CVAL;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        return CNTV_CVAL;
elseif PSTATE.EL == EL2 then
    return CNTV_CVAL;
elseif PSTATE.EL == EL3 then
    return CNTV_CVAL;

```

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0011 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elseif ELUsingAArch32(EL1) && CNTKCTL_EL1.EL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = R[t2]:R[t];
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CVAL_EL2 = R[t2]:R[t];
    else
        CNTV_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        CNTV_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL2 then
    CNTV_CVAL = R[t2]:R[t];
elseif PSTATE.EL == EL3 then
    CNTV_CVAL = R[t2]:R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_TVAL, Counter-timer Virtual Timer TimerValue register

The CNTV_TVAL characteristics are:

Purpose

Holds the timer value for the virtual timer.

Configuration

AArch32 System register CNTV_TVAL bits [31:0] are architecturally mapped to AArch64 System register [CNTV_TVAL_EL0\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTV_TVAL are UNDEFINED.

Attributes

CNTV_TVAL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TimerValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

TimerValue, bits [31:0]

The TimerValue view of the virtual timer.

On a read of this register:

- If [CNTV_CTL.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTV_CTL.ENABLE](#) is 1, the value returned is ([CNTV_CVAL](#) - [CNTVCT](#)).

On a write of this register, [CNTV_CVAL](#) is set to ([CNTVCT](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTP_CTL.ENABLE](#) is 1, the timer condition is met when ([CNTVCT](#) - [CNTP_CVAL](#)) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTV_CTL.ISTATUS](#) is set to 1.
- If [CNTV_CTL.IMASK](#) is 0, an interrupt is generated.

When [CNTV_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTVCT](#) continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_TVAL

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b000 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        if CNTHVS_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTHVS_CVAL_EL2 - PhysicalCountInt())<31:0>;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        if CNTHV_CTL_EL2.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTHV_CVAL_EL2 - PhysicalCountInt())<31:0>;
    else
        if CNTV_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        elsif HaveEL(EL2) && !ELUsingAArch32(EL2) then
            return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>;
        elsif HaveEL(EL2) && ELUsingAArch32(EL2) then
            return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
        else
            return (CNTV_CVAL - PhysicalCountInt())<31:0>;
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            if CNTV_CTL.ENABLE == '0' then
                return bits(32) UNKNOWN;
            elsif HaveEL(EL2) && !ELUsingAArch32(EL2) then
                return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF_EL2))<31:0>;
            elsif HaveEL(EL2) && ELUsingAArch32(EL2) then
                return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
            else
                return (CNTV_CVAL - PhysicalCountInt())<31:0>;
    elsif PSTATE.EL == EL2 then
        if CNTV_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        else
            return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
    elsif PSTATE.EL == EL3 then
        if CNTV_CTL.ENABLE == '0' then
            return bits(32) UNKNOWN;
        elsif HaveEL(EL2) then
            return (CNTV_CVAL - (PhysicalCountInt() - CNTVOFF))<31:0>;
        else
            return (CNTV_CVAL - PhysicalCountInt())<31:0>;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CNTKCTL_EL1.EL0VTEN
== '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && CNTKCTL.PL0VTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CNTHCTL_EL2.EL0VTEN
== '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '0'
&& IsFeatureImplemented(FEAT_SEL2) then
        CNTHVS_CVAL_EL2 = SignExtend(R[t],64) + PhysicalCountInt();
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCR_EL3.NS == '1'
then
        CNTHV_CVAL_EL2 = SignExtend(R[t],64) + PhysicalCountInt();
    else
        if HaveEL(EL2) && !ELUsingAArch32(EL2) then
            CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt() - CNTVOFF_EL2;
        elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
            CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt() - CNTVOFF;
        else
            CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVT == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            if HaveEL(EL2) && !ELUsingAArch32(EL2) then
                CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt() - CNTVOFF_EL2;
            elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
                CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt() - CNTVOFF;
            else
                CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt();
    elseif PSTATE.EL == EL2 then
        CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt() - CNTVOFF;
    elseif PSTATE.EL == EL3 then
        if HaveEL(EL2) then
            CNTV_CVAL = SignExtend(R[t], 64) + PhysicalCountInt() - CNTVOFF;
        else
            CNTV_CVAL = SignExtend(R[t],64) + PhysicalCountInt();

```

CNTVCT, Counter-timer Virtual Count register

The CNTVCT characteristics are:

Purpose

Holds the 64-bit virtual count value. The virtual count value is equal to the physical count value minus the virtual offset visible in [CNTVOFF](#).

Configuration

AArch32 System register CNTVCT bits [63:0] are architecturally mapped to AArch64 System register [CNTVCT_ELO\[63:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTVCT are UNDEFINED.

The value of this register is the same as the value of [CNTPCT](#) in the following conditions:

- When EL2 is not implemented.
- When EL2 is implemented and is using AArch64, [HCR_EL2](#).{E2H, TGE} is {1, 1}, and this register is read from Non-secure EL0.

All reads to the CNTVCT occur in program order relative to reads to [CNTVCTSS](#) or CNTVCT.

Attributes

CNTVCT is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual count value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Virtual count value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual count value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVCT

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
CNTKCTL_EL1.EL0VCTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elseif ELUsingAArch32(EL1) && CNTKCTL_PL0VCTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
CNTHCTL_EL2.EL0VCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVCT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        if HaveEL(EL2) && !ELUsingAArch32(EL2) && (!EL2Enabled() || HCR_EL2.<E2H,TGE> != '11') then
            return PhysicalCountInt() - CNTV0FF_EL2;
        elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
            return PhysicalCountInt() - CNTV0FF;
        else
            return PhysicalCountInt();
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVCT == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        if HaveEL(EL2) && !ELUsingAArch32(EL2) then
            return PhysicalCountInt() - CNTV0FF_EL2;
        elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
            return PhysicalCountInt() - CNTV0FF;
        else
            return PhysicalCountInt();
elseif PSTATE.EL == EL2 then
    return PhysicalCountInt() - CNTV0FF;
elseif PSTATE.EL == EL3 then
    if HaveEL(EL2) then
        return PhysicalCountInt() - CNTV0FF;
    else
        return PhysicalCountInt();

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVCTSS, Counter-timer Self-Synchronized Virtual Count register

The CNTVCTSS characteristics are:

Purpose

Holds the 64-bit virtual count value. The virtual count value is equal to the physical count value visible in [CNTPCT](#) minus the virtual offset visible in [CNTVOFF](#).

Configuration

AArch32 System register CNTVCTSS bits [63:0] are architecturally mapped to AArch64 System register [CNTVCTSS_EL0\[63:0\]](#).

This register is present only when AArch32 is supported and FEAT_ECV is implemented. Otherwise, direct accesses to CNTVCTSS are UNDEFINED.

All reads to the CNTVCTSS occur in program order relative to reads to [CNTVCT](#) or CNTVCTSS.

This register is a self-synchronised view of the [CNTVCT](#) counter, and cannot be read speculatively.

Attributes

CNTVCTSS is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Self-Synchronized Virtual count value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Self-Synchronized Virtual count value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVCTSS

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b1001 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') &&
CNTKCTL_EL1.EL0VCTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elseif ELUsingAArch32(EL1) && CNTKCTL_PL0VCTEN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' &&
CNTHCTL_EL2.EL0VCTEN == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && CNTHCTL_EL2.EL1TVCT
== '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    else
        if HaveEL(EL2) && !ELUsingAArch32(EL2) && (!EL2Enabled() || HCR_EL2.<E2H,TGE> != '11') then
            return PhysicalCountInt() - CNTV0FF_EL2;
        elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
            return PhysicalCountInt() - CNTV0FF;
        else
            return PhysicalCountInt();
    elseif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && CNTHCTL_EL2.EL1TVCT == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            if HaveEL(EL2) && !ELUsingAArch32(EL2) then
                return PhysicalCountInt() - CNTV0FF_EL2;
            elseif HaveEL(EL2) && ELUsingAArch32(EL2) then
                return PhysicalCountInt() - CNTV0FF;
            else
                return PhysicalCountInt();
    elseif PSTATE.EL == EL2 then
        return PhysicalCountInt() - CNTV0FF;
    elseif PSTATE.EL == EL3 then
        if HaveEL(EL2) then
            return PhysicalCountInt() - CNTV0FF;
        else
            return PhysicalCountInt();

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVOFF, Counter-timer Virtual Offset register

The CNTVOFF characteristics are:

Purpose

Holds the 64-bit virtual offset. This is the offset between the physical count value visible in [CNTPCT](#) and the virtual count value visible in [CNTVCT](#).

Configuration

AArch32 System register CNTVOFF bits [63:0] are architecturally mapped to AArch64 System register [CNTVOFF_EL2\[63:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CNTVOFF are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3 and the virtual counter uses a fixed virtual offset of zero.

Note

When EL2 is implemented and is using AArch64, if [HCR_EL2](#).{E2H, TGE} is {1, 1}, the virtual counter uses a fixed virtual offset of zero when [CNTVCT](#) is read from Non-secure EL0.

Attributes

CNTVOFF is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Virtual offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual offset.

If the Generic counter is implemented at a size less than 64 bits, then this field is permitted to be implemented at the same width as the counter, and the upper bits are RES0.

The value of this field is treated as zero-extended in all counter calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVOFF

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    return CNTVOFF;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return CNTVOFF;

```

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b1110 | 0b0100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    UNDEFINED;
elsif PSTATE.EL == EL2 then
    CNTVOFF = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        CNTVOFF = R[t2]:R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CONTEXTIDR, Context ID Register

The CONTEXTIDR characteristics are:

Purpose

Identifies the current Process Identifier and, when using the Short-descriptor translation table format, the Address Space Identifier.

The value of the whole of this register is called the Context ID and is used by:

- The debug logic, for Linked and Unlinked Context ID matching.
- The trace logic, to identify the current process.

The significance of this register is for debug and trace use only.

Configuration

AArch32 System register CONTEXTIDR bits [31:0] are architecturally mapped to AArch64 System register [CONTEXTIDR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CONTEXTIDR are UNDEFINED.

The register format depends on whether address translation is using the Long-descriptor or the Short-descriptor translation table format.

Attributes

CONTEXTIDR is a 32-bit register.

Field descriptions

When TTBCR.EAE == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PROCID | | | | | | | | | | | | | | | | | | | | | | | | ASID | | | | | | | |

PROCID, bits [31:8]

Process Identifier. This field must be programmed with a unique value that identifies the current process.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ASID, bits [7:0]

Address Space Identifier. This field is programmed with the value of the current ASID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When TTBCR.EAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PROCID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

PROCID, bits [31:0]

Process Identifier. This field must be programmed with a unique value that identifies the current process.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CONTEXTIDR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CONTEXTIDR_NS;
    else
        return CONTEXTIDR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CONTEXTIDR_NS;
    else
        return CONTEXTIDR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return CONTEXTIDR_S;
    else
        return CONTEXTIDR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        CONTEXTIDR_NS = R[t];
    else
        CONTEXTIDR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CONTEXTIDR_NS = R[t];
    else
        CONTEXTIDR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CONTEXTIDR_S = R[t];
    else
        CONTEXTIDR_NS = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CP15DMB, Data Memory Barrier System instruction

The CP15DMB characteristics are:

Purpose

Performs a Data Memory Barrier.

Arm deprecates any use of this System instruction, and strongly recommends that software use the DMB instruction instead.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to CP15DMB are UNDEFINED.

Attributes

CP15DMB is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the CP15DMB instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1010 | 0b101 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.CP15BEN
== '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.CP15BEN
== '0' then
        UNDEFINED;
    elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        CP15DMB();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DMB();
elsif PSTATE.EL == EL2 then
    if HSCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DMB();
elsif PSTATE.EL == EL3 then
    if SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DMB();

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CP15DSB, Data Synchronization Barrier System instruction

The CP15DSB characteristics are:

Purpose

Performs a Data Synchronization Barrier.

Arm deprecates any use of this System instruction, and strongly recommends that software use the DSB instruction instead.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to CP15DSB are UNDEFINED.

Attributes

CP15DSB is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the CP15DSB instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1010 | 0b100 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.CP15BEN
== '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.CP15BEN
== '0' then
        UNDEFINED;
    elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        CP15DSB();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DSB();
elsif PSTATE.EL == EL2 then
    if HSCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DSB();
elsif PSTATE.EL == EL3 then
    if SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15DSB();

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CP15ISB, Instruction Synchronization Barrier System instruction

The CP15ISB characteristics are:

Purpose

Performs an Instruction Synchronization Barrier.

Arm deprecates any use of this System instruction, and strongly recommends that software use the ISB instruction instead.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to CP15ISB are UNDEFINED.

Attributes

CP15ISB is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the CP15ISB instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b0101 | 0b100 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.CP15BEN
== '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.CP15BEN
== '0' then
        UNDEFINED;
    elsif ELUsingAArch32(EL1) && SCTLR.CP15BEN == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        CP15ISB();
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15ISB();
elsif PSTATE.EL == EL2 then
    if HSCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15ISB();
elsif PSTATE.EL == EL3 then
    if SCTLR.CP15BEN == '0' then
        UNDEFINED;
    else
        CP15ISB();

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPACR, Architectural Feature Access Control Register

The CPACR characteristics are:

Purpose

Controls access to trace, and to Advanced SIMD and floating-point functionality from EL0, EL1, and EL3.

In an implementation that includes EL2, the CPACR has no effect on instructions executed at EL2.

Configuration

AArch32 System register CPACR bits [31:0] are architecturally mapped to AArch64 System register [CPACR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CPACR are UNDEFINED.

Bits in the [NSACR](#) control Non-secure access to the CPACR fields. See the field descriptions for more information.

Note

In the register field descriptions, controls are described as applying at specified Privilege levels. This is because, in Secure state, a PL1 control:

- Applies to execution in a Secure EL3 mode when EL3 is using AArch32.
- Applies to execution in a Secure EL1 mode when EL3 is using AArch64.

See 'Security state, Exception levels, and AArch32 execution privilege'.

Attributes

CPACR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|------|--------|----|------|----|----|----|------|------|----|----|----|----|----|----|----|----|----|----|----|----|------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ASEDIS | RES0 | TRCDIS | | RES0 | | | | cp11 | cp10 | | | | | | | | | | | | | RES0 | | | | | | | | | |

ASEDIS, bit [31]

Disables PL0 and PL1 execution of Advanced SIMD instructions.

| ASEDIS | Meaning |
|--------|--|
| 0b0 | This control permits execution of Advanced SIMD instructions at PL0 and PL1. |
| 0b1 | All instruction encodings that are Advanced SIMD instruction encodings, but are not also floating-point instruction encodings, are UNDEFINED at PL0 and PL1. |

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0. Otherwise, it is IMPLEMENTATION DEFINED whether this field is implemented as a RW field. If it is not implemented as a RW field, it is RAZ/WI.

If EL3 is implemented and is using AArch32, and the value of [NSACR.NSASEDIS](#) is 1, this field behaves as RAO/WI in Non-secure state, regardless of its actual value. This applies even if the field is implemented as RAZ/WI.

For the list of instructions affected by this field, see 'Controls of Advanced SIMD operation that do not apply to floating-point operation'.

See the description of CPACR.cp10 for a list of other controls that can disable or trap execution of Advanced SIMD instructions in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [30:29]

Reserved, RES0.

TRCDIS, bit [28]

Traps PL0 and PL1 System register accesses to all implemented trace registers to Undefined mode.

| TRCDIS | Meaning |
|--------|--|
| 0b0 | This control has no effect on PL0 and PL1 System register accesses to trace registers. |
| 0b1 | PL0 and PL1 System register accesses to all implemented trace registers are trapped to Undefined mode. |

If the implementation does not include a PE trace unit, or does not include a System register interface to the PE trace unit registers, this field is RES0. Otherwise, it is IMPLEMENTATION DEFINED whether this field is implemented as a RW field. If it is not implemented as a RW field, it is RAZ/WI.

If EL3 is implemented and is using AArch32, and the value of [NSACR.NSTRCDIS](#) is 1, this field behaves as RAO/WI in Non-secure state, regardless of its actual value. This applies even if the field is implemented as RAZ/WI.

Note

- The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the PE trace unit implements FEAT_ETMv4 or FEAT_ETE, EL0 accesses to the trace registers are UNDEFINED.
- The Arm architecture does not provide traps on trace register accesses through the optional memory-mapped external debug interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [27:24]

Reserved, RES0.

cp11, bits [23:22]

The value of this field is ignored. If this field is programmed with a different value to the cp10 field then this field is UNKNOWN on a direct read of the CPACR.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

In Non-secure state, if EL3 is implemented and is using AArch32, when the value of [NSACR.cp10](#) is 0, this field behaves as RAZ/WI, regardless of its actual value.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

cp10, bits [21:20]

Defines the access rights for the Advanced SIMD and floating-point functionality. Possible values of the field are:

| cp10 | Meaning |
|------|---|
| 0b00 | PL0 and PL1 accesses to Advanced SIMD and floating-point registers or instructions are UNDEFINED. |
| 0b01 | PL0 accesses to Advanced SIMD and floating-point registers or instructions are UNDEFINED. |
| 0b10 | Reserved. The effect of programming this field to this value is CONSTRAINED UNPREDICTABLE. See 'Handling of System register control fields for Advanced SIMD and floating-point operation'. |
| 0b11 | This control permits full access to the Advanced SIMD and floating-point functionality from PL0 and PL1. |

The Advanced SIMD and floating-point features controlled by these fields are:

- Execution of any floating-point or Advanced SIMD instruction.
- Any access to the Advanced SIMD and floating-point registers D0-D31 and their views as S0-S31 and Q0-Q15.
- Any access to the [FPSCR](#), [FPSID](#), [MVFR0](#), [MVFR1](#), [MVFR2](#), or [FPEXC](#) System registers.

Note

The [CPACR](#) has no effect on Advanced SIMD and floating-point accesses from PL2. These can be disabled by the [HCPTR](#).TCP10 field.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

In Non-secure state, if EL3 is implemented and is using AArch32, when the value of [NSACR](#).cp10 is 0, this field behaves as RAZ/WI, regardless of its actual value.

Execution of Advanced SIMD and floating-point instructions in AArch32 state can be disabled or trapped by the following controls:

- CPACR.cp10, or, if executing at EL0, [CPACR_EL1](#).FPEN.
- [FPEXC](#).EN.
- If executing in Non-secure state:
 - [HCPTR](#).TCP10, or if EL2 is using AArch64, [CPTR_EL2](#).TFP.
 - [NSACR](#).cp10, or if EL3 is using AArch64, [CPTR_EL3](#).TFP.
- For Advanced SIMD instructions only:
 - CPACR.ASEDIS.
 - If executing in Non-secure state, [HCPTR](#).TASE and [NSACR](#).NSTRCDIS.

See the descriptions of the controls for more information.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [19:0]

Reserved, RES0.

Accessing CPACR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TCPAC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return CPACR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            else
                return CPACR;
    elsif PSTATE.EL == EL3 then
        return CPACR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && CPTR_EL2.TCPAC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCPTR.TCPAC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            CPACR = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            else
                CPACR = R[t];
    elsif PSTATE.EL == EL3 then
        CPACR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPPRCTX, Cache Prefetch Prediction Restriction by Context

The CPPRCTX characteristics are:

Purpose

Cache Prefetch Prediction Restriction by Context applies to all Cache Allocation Resources that predict cache allocations based on information gathered within the target execution context or contexts.

Cache prefetch predictions determined by the actions of code in the target execution context or contexts appearing in program order before the instruction cannot influence speculative execution occurring after the instruction is complete and synchronized.

This instruction applies to all:

- Instruction caches.
- Data caches.
- TLB prefetching hardware used by the executing PE that applies to the supplied context or contexts.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of Cache Allocation Resources so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

Configuration

This instruction is present only when AArch32 is supported and FEAT_SPECRES is implemented. Otherwise, direct accesses to CPPRCTX are UNDEFINED.

Attributes

CPPRCTX is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|---------|----|----|----|----|----|----|----|------|----|----|----|------|----|----|----|-------|----|---|---|------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | GVMIDNS | | | | EL | | | | VMID | | | | RES0 | | | | GASID | | | | ASID | | | | | | | |

Bits [31:28]

Reserved, RES0.

GVMID, bit [27]

Execution of this instruction applies to all VMIDs or a specified VMID.

| GVMID | Meaning |
|--------------|---|
| 0b0 | Applies to specified VMID for an EL0 or EL1 target execution context. |
| 0b1 | Applies to all VMIDs for an EL0 or EL1 target execution context. |

For target execution contexts other than EL0 or EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, then this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

NS, bit [26]

Security State.

| NS | Meaning |
|-----------|-------------------|
| 0b0 | Secure state. |
| 0b1 | Non-secure state. |

If the instruction is executed in Non-secure state, this field is treated as 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

| EL | Meaning |
|-----------|----------------|
| 0b00 | EL0. |
| 0b01 | EL1. |
| 0b10 | EL2. |
| 0b11 | EL3. |

If the instruction is executed at an Exception level lower than the specified level, this instruction is treated as a NOP.

VMID, bits [23:16]

Only applies when bit[27] is 0 and the target execution context is either:

- EL1.
- EL0 when ([HCR_EL2.E2H==0](#) or [HCR_EL2.TGE==0](#)) or EL2 is using AArch32 state.

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0 and ([HCR_EL2.E2H==0](#) or [HCR_EL2.TGE==0](#) or [ELUsingAArch32\(EL2\)](#)), this field is treated as the current VMID.

When the instruction is executed at EL0 and ([HCR_EL2.E2H==1](#) and [HCR_EL2.TGE==1](#) and [!ELUsingAArch32\(EL2\)](#)), this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

Bits [15:9]

Reserved, RES0.

GASID, bit [8]

Execution of this instruction applies to all ASIDs or a specified ASID.

| GASID | Meaning |
|--------------|--|
| 0b0 | Applies to specified ASID for an EL0 target execution context. |
| 0b1 | Applies to all ASID for an EL0 target execution context. |

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field has an Effective value of 0.

ASID, bits [7:0]

Only applies for an EL0 target execution context and when bit[8] is 0.

Otherwise, this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

Executing the CPPRCTX instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b0111 | 0b0011 | 0b111 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTL_EL1.EnRCTX ==
'0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && SCTL_EL1.EnRCTX == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.CPPRCTX == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTL_EL2.EnRCTX ==
'0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        AArch32.RestrictPrediction(R[t], RestrictType_CachePrefetch);
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.NV == '1' then
        AArch64.SystemAccessTrap(EL2, 0x03);
    else
        AArch32.RestrictPrediction(R[t], RestrictType_CachePrefetch);
elseif PSTATE.EL == EL2 then
    AArch32.RestrictPrediction(R[t], RestrictType_CachePrefetch);
elseif PSTATE.EL == EL3 then
    AArch32.RestrictPrediction(R[t], RestrictType_CachePrefetch);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CPSR, Current Program Status Register

The CPSR characteristics are:

Purpose

Holds PE status and control information.

Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to CPSR are UNDEFINED.

Attributes

CPSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|------|------|-----|-----|------|----|------|----|----|----|----|------|------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| N | Z | C | V | Q | RES0 | SSBS | PAN | DIT | RES0 | GE | RES0 | E | A | I | F | RES0 | RES1 | M | | | | | | | | | | | | | |

N, bit [31]

Negative condition flag. Set to bit[31] of the result of the last flag-setting instruction. If the result is regarded as a two's complement signed integer, then N is set to 1 if the result was negative, and N is set to 0 if the result was positive or zero.

Z, bit [30]

Zero condition flag. Set to 1 if the result of the last flag-setting instruction was zero, and to 0 otherwise. A result of zero often indicates an equal result from a comparison.

C, bit [29]

Carry condition flag. Set to 1 if the last flag-setting instruction resulted in a carry condition, for example an unsigned overflow on an addition.

V, bit [28]

Overflow condition flag. Set to 1 if the last flag-setting instruction resulted in an overflow condition, for example a signed overflow on an addition.

Q, bit [27]

Cumulative saturation bit. Set to 1 to indicate that overflow or saturation occurred in some instructions.

Bits [26:24]

Reserved, RES0.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass Safe.

Prohibits speculative loads or stores that might practically allow a cache timing side channel.

A cache timing side channel might be exploited where a load or store uses an address that is derived from a register that is being loaded from memory using a load instruction speculatively read from a memory location. If PSTATE.SSBS is enabled, the address derived from the load instruction might be from earlier in the coherence order than the latest store to that memory location with the same virtual address.

| SSBS | Meaning |
|------|---|
| 0b0 | Hardware is not permitted to load or store speculatively in the manner described. |
| 0b1 | Hardware is permitted to load or store speculatively in the manner described. |

The value of this bit is usually set to the value described by the [SCTLR.DSSBS](#) bit on exceptions to any mode except Hyp mode, and the value described by [HSTCLR.DSSBS](#) on exceptions to Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never.

| PAN | Meaning |
|-----|---|
| 0b0 | The translation system is the same as Armv8.0. |
| 0b1 | Disables privileged read and write accesses to addresses accessible at EL0. |

The value of this bit is usually preserved on taking an exception, except in the following situations:

- When the target of the exception is EL1, and the value of the [SCTLR.SPAN](#) bit for the current Security state is 0, this bit is set to 1.
- When the target of the exception is EL3, from Secure state, and the value of the Secure [SCTLR.SPAN](#) is 0, this bit is set to 1.
- When the target of the exception is EL3, from Non-secure state, this bit is set to 0 regardless of the value of the Secure [SCTLR.SPAN](#) bit.

Otherwise:

Reserved, RES0.

DIT, bit [21]

When FEAT_DIT is implemented:

Data Independent Timing.

| DIT | Meaning |
|------------|---|
| 0b0 | The architecture makes no statement about the timing properties of any instructions. |
| 0b1 | <p>The architecture requires that:</p> <ul style="list-style-type: none"> • The timing of every load and store instruction is insensitive to the value of the data being loaded or stored. • For certain data processing instructions, the instruction takes a time that is independent of: <ul style="list-style-type: none"> ◦ The values of the data supplied in any of its registers. ◦ The values of the NZCV flags. • For certain data processing instructions, the response of the instruction to asynchronous exceptions does not vary based on: <ul style="list-style-type: none"> ◦ The values of the data supplied in any of its registers. ◦ The values of the NZCV flags. |

The data processing instructions affected by this bit are:

- All cryptographic instructions. These instructions are:
 - AESD, AESE, AESIMC, AESMC, SHA1C, SHA1H, SHA1M, SHA1P, SHA1SU0, SHA1SU1, SHA256H, SHA256H2, SHA256SU0, and SHA256SU1.
- A subset of the instructions that use the general-purpose register file. For these instructions, the effects of CPSR.DIT apply only if they do not use R15 as either their source or destination and pass their condition execution check. These instructions are:
 - BFI, BFC, CLZ, CMN, CMP, MLA, MLAS, MLS, MOVT, MUL, MULS, NOP, PKHBT, PKHTB, RBIT, REV, REV16, REVSH, RRX, SADD16, SADD8, SASX, SBFX, SHADD16, SHADD8, SHASX, SHSAX, SHSUB16, SHSUB8, SMLAL**, SMLAW*, SMLSD*, SMLLA*, SMLLS*, SMMUL*, SMUAD*, SMUL*, SSAX, SSUB16, SSUB8, SXTAB*, SXTAH, SXTB*, SXTH, TEQ, TST, UADD*, UASX, UBFX, UHADD*, UHASX, UHSAX, UHSUB*, UMAAL, UMLAL, UMLALS, UMULL, UMULLS, USADA8, USAX, USUB*, UXTAB*, UXTAH, UXTB*, UXTH, ADC (register-shifted register), ADCS (register-shifted register), ADD (register-shifted register), ADDS (register-shifted register), AND (register-shifted register), ANDS (register-shifted register), ASR (register-shifted register), ASRS (register-shifted register), BIC (register-shifted register), BICS (register-shifted register), EOR (register-shifted register), EORS (register-shifted register), LSL (register-shifted register), LSLS (register-shifted register), LSR (register-shifted register), LSRS (register-shifted register), MOV (register-shifted register), MOVS (register-shifted register), MVN (register-shifted register), MVNS (register-shifted register), ORR (register-shifted register), ORRS (register-shifted register), ROR (register-shifted register), RORS (register-shifted register), RSB (register-shifted register), RSBS (register-shifted register), RSC (register-shifted register), RSCS (register-shifted register), SBC (register-shifted register), SBCS (register-shifted register), SUB (register-shifted register), and SUBS (register-shifted register).
- A subset of the instructions that use the general-purpose register file. For these instructions, the effects of CPSR.DIT do not depend on these instructions passing their condition execution check. These instructions are:
 - ADC (immediate), ADC (register), ADCS (immediate), ADCS (register), ADD (immediate), ADD (register), ADDS (immediate), ADDS (register), AND (immediate), AND (register), ANDS (immediate), ANDS (register), ASR (immediate), ASR (register), ASRS (immediate), ASRS (register), BIC (immediate), BIC (register), BICS (immediate), BICS (register), EOR (immediate), EOR (register), EORS (immediate), EORS (register), LSL (immediate), LSL (register), LSLS (immediate), LSLS (register), LSR (immediate), LSR (register), LSRS (immediate), LSRS (register), MOV (immediate), MOV (register), MOVS (immediate), MOVS (register), MVN (immediate), MVN (register), MVNS (immediate), MVNS (register), ORR (immediate), ORR (register), ORRS (immediate), ORRS (register), ROR (immediate), ROR (register), RORS (immediate), RORS (register), RSB (immediate), RSB (register), RSBS (immediate), RSBS (register), RSC (immediate), RSC (register), RSCS (immediate), RSCS (register), SBC (immediate), SBC (register), SBCS (immediate), SBCS (register), SUB (immediate), SUB (register), SUBS (immediate), and SUBS (register).
 - If FEAT_CRC32 is implemented, CRC32B, CRC32H, CRC32W, CRC32CB, CRC32CH, and CRC32CW.
- A subset of the instructions that use the SIMD&FP register file. For these instructions, the effects of CPSR.DIT apply only if they pass their condition execution check. These instructions are:

- VABA*, VABD* (integer), VADD (integer), VADDHN, VADDL, VADDW, VAND, VBIC, VBIF, VBIT, VBSL, VCLS, VCLZ, VCNT, VDUP, VEOR, VEXT, VHADD, VHSUB, VMAX (integer), VMIN (integer), VMLA (integer), VMLAL, VMLS (integer), VMLS, VMOV, VMOVL, VMOVN, VMUL (integer and polynomial), VMULL (integer and polynomial), VMVN, VORN, VORR, VPADAL, VPADD (integer), VPADDL, VPMAX (integer), VPMIN (integer), VRADDHN, VREV*, VRHADD, VRSHL, VRSHR, VRSRN, VRSRA, VRSUBHN, VSHL, VSHLL, VSHR, VSLI, VSRA, VSRI, VSUB (integer), VSUBHN, VSUBL, VSUBW, VSWP, VTBL, VTBX, VTRN, VTST, VUZP, and VZIP.
- Another subset of the instructions that use the SIMD&FP register file. For these instructions, the effects of CPSR.DIT apply only if they pass their condition execution check and apply only when the instructions are operating on integer vector elements. These instructions are:
 - VABS, VCGE, VCGT, VCLE, VCLT, VMLA (by scalar), VMLS (by scalar), and VNEG.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [20]

Reserved, RES0.

GE, bits [19:16]

Greater than or Equal flags, for parallel addition and subtraction.

Bits [15:10]

Reserved, RES0.

E, bit [9]

Endianness state bit. Controls the load and store endianness for data accesses:

| E | Meaning |
|-----|-------------------------|
| 0b0 | Little-endian operation |
| 0b1 | Big-endian operation. |

Instruction fetches ignore this bit.

If an implementation does not provide Big-endian support, this bit is RES0. If it does not provide Little-endian support, this bit is RES1.

If an implementation provides Big-endian support but only at EL0, this bit is RES0 for an exception return to any Exception level other than EL0.

Likewise, if it provides Little-endian support only at EL0, this bit is RES1 for an exception return to any Exception level other than EL0.

When the reset value of the SCTLR.EE bit is defined by a configuration input signal, that value also applies to the CPSR.E bit on reset, and therefore applies to software execution from reset.

A, bit [8]

SError interrupt mask bit.

| A | Meaning |
|-----|-----------------------|
| 0b0 | Exception not masked. |
| 0b1 | Exception masked. |

I, bit [7]

IRQ mask bit.

| I | Meaning |
|----------|-----------------------|
| 0b0 | Exception not masked. |
| 0b1 | Exception masked. |

F, bit [6]

FIQ mask bit.

| F | Meaning |
|----------|-----------------------|
| 0b0 | Exception not masked. |
| 0b1 | Exception masked. |

Bit [5]

Reserved, RES0.

Bit [4]

Reserved, RES1.

M, bits [3:0]

Current PE mode.

| M | Meaning |
|----------|----------------|
| 0b0000 | User. |
| 0b0001 | FIQ. |
| 0b0010 | IRQ. |
| 0b0011 | Supervisor. |
| 0b0110 | Monitor. |
| 0b0111 | Abort. |
| 0b1010 | Hyp. |
| 0b1011 | Undefined. |
| 0b1111 | System. |

Accessing CPSR

CPSR can be read using the MRS instruction and written using the MSR (register) or MSR (immediate) instructions.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CSSELR, Cache Size Selection Register

The CSSELR characteristics are:

Purpose

Selects the current Cache Size ID Register, [CCSIDR](#), by specifying the required cache level and the cache type, which is either instruction cache or data cache.

If FEAT_CCIDX is implemented, CSSELR also selects the current [CCSIDR2](#).

Configuration

AArch32 System register CSSELR bits [31:0] are architecturally mapped to AArch64 System register [CSSELR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CSSELR are UNDEFINED.

Attributes

CSSELR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|-----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | Level | | | InD | | | | | | | | | | | | |

Bits [31:4]

Reserved, RES0.

Level, bits [3:1]

Cache level of required cache. Permitted values are:

| Level | Meaning |
|-------|----------------|
| 0b000 | Level 1 cache. |
| 0b001 | Level 2 cache. |
| 0b010 | Level 3 cache. |
| 0b011 | Level 4 cache. |
| 0b100 | Level 5 cache. |
| 0b101 | Level 6 cache. |
| 0b110 | Level 7 cache. |

All other values are reserved.

If CSSELR.Level is programmed to a cache level that is not implemented, then the value for this field on a read of CSSELR is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

InD, bit [0]

Instruction not Data bit. Permitted values are:

| InD | Meaning |
|-----|------------------------|
| 0b0 | Data or unified cache. |
| 0b1 | Instruction cache. |

If CSSELR.Level is programmed to a cache level that is not implemented, then the value for this field on a read of CSSELR is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing CSSELR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b010 | 0b0000 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CSSELR_NS;
    else
        return CSSELR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return CSSELR_NS;
    else
        return CSSELR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return CSSELR_S;
    else
        return CSSELR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b010 | 0b0000 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID4 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TID4 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        CSSELR_NS = R[t];
    else
        CSSELR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        CSSELR_NS = R[t];
    else
        CSSELR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        CSSELR_S = R[t];
    else
        CSSELR_NS = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTR, Cache Type Register

The CTR characteristics are:

Purpose

Provides information about the architecture of the caches.

Configuration

AArch32 System register CTR bits [31:0] are architecturally mapped to AArch64 System register [CTR_EL0\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to CTR are UNDEFINED.

Attributes

CTR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|-----|-----|-----|----|----|----|-----|----|----|----|----------|----|----|----|------|----|----|----|------|----|---|---|---|---|---|---|----------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES1 | RES0 | DIC | IDC | CWG | | | | ERG | | | | DminLine | | | | L1lp | | | | RES0 | | | | | | | | lminLine | | | |

Bit [31]

Reserved, RES1.

Bit [30]

Reserved, RES0.

DIC, bit [29]

Instruction cache invalidation requirements for data to instruction coherence.

| DIC | Meaning |
|-----|---|
| 0b0 | Instruction cache invalidation to the Point of Unification is required for data to instruction coherence. |
| 0b1 | Instruction cache invalidation to the Point of Unification is not required for data to instruction coherence. |

IDC, bit [28]

Data cache clean requirements for instruction to data coherence. The meaning of this bit is:

| IDC | Meaning |
|-----|---|
| 0b0 | Data cache clean to the Point of Unification is required for instruction to data coherence, unless CLIDR.LoC == 0b000 or (CLIDR.LoUIS == 0b000 && CLIDR.LoUU == 0b000). |
| 0b1 | Data cache clean to the Point of Unification is not required for instruction to data coherence. |

CWG, bits [27:24]

Cache writeback granule. Log₂ of the number of words of the maximum size of memory that can be overwritten as a result of the eviction of a cache entry that has had a memory location in it modified.

A value of 0b0000 indicates that this register does not provide Cache writeback granule information and either:

- The architectural maximum of 512 words (2KB) must be assumed.
- The Cache writeback granule can be determined from maximum cache line size encoded in the Cache Size ID Registers.

Values greater than 0b1001 are reserved.

Arm recommends that an implementation that does not support cache write-back implements this field as 0b0001. This applies, for example, to an implementation that supports only write-through caches.

ERG, bits [23:20]

Exclusives reservation granule. Log₂ of the number of words of the maximum size of the reservation granule that has been implemented for the Load-Exclusive and Store-Exclusive instructions.

The use of the value 0b0000 is deprecated.

The value 0b0001 and values greater than 0b1001 are reserved.

DminLine, bits [19:16]

Log₂ of the number of words in the smallest cache line of all the data caches and unified caches that are controlled by the PE.

L1Ip, bits [15:14]

Level 1 instruction cache policy. Indicates the indexing and tagging policy for the L1 instruction cache. Possible values of this field are:

| L1Ip | Meaning | Applies when |
|------|--|--------------------------------|
| 0b00 | VMID aware Physical Index, Physical tag (VPIPT). | When FEAT_VPIPT is implemented |
| 0b01 | ASID-tagged Virtual Index, Virtual Tag (AIVIVT). | |
| 0b10 | Virtual Index, Physical Tag (VIPT). | |
| 0b11 | Physical Index, Physical Tag (PIPT). | |

The value 0b00 is permitted only in an implementation that includes FEAT_VPIPT, otherwise the value is reserved.

The value 0b01 is not permitted in Armv8.

Bits [13:4]

Reserved, RES0.

IminLine, bits [3:0]

Log₂ of the number of words in the smallest cache line of all the instruction caches that are controlled by the PE.

Accessing CTR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return CTR;
elsif PSTATE.EL == EL2 then
    return CTR;
elsif PSTATE.EL == EL3 then
    return CTR;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DACR, Domain Access Control Register

The DACR characteristics are:

Purpose

Defines the access permission for each of the sixteen memory domains.

Configuration

AArch32 System register DACR bits [31:0] are architecturally mapped to AArch64 System register [DACR32_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DACR are UNDEFINED.

This register has no function when [TTBCR](#).EAE is set to 1, to select the Long-descriptor translation table format.

Attributes

DACR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| D15 | D14 | D13 | D12 | D11 | D10 | D9 | D8 | D7 | D6 | D5 | D4 | D3 | D2 | D1 | D0 | | | | | | | | | | | | | | | | |

D<n>, bits [2n+1:2n], for n = 15 to 0

Domain n access permission, where n = 0 to 15. Permitted values are:

| D<n> | Meaning |
|------|--|
| 0b00 | No access. Any access to the domain generates a Domain fault. |
| 0b01 | Client. Accesses are checked against the permission bits in the translation tables. |
| 0b11 | Manager. Accesses are not checked against the permission bits in the translation tables. |

The value 0b10 is reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DACR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0011 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return DACR_NS;
    else
        return DACR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return DACR_NS;
    else
        return DACR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return DACR_S;
    else
        return DACR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0011 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        DACR_NS = R[t];
    else
        DACR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        DACR_NS = R[t];
    else
        DACR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            DACR_S = R[t];
        else
            DACR_NS = R[t];

```


DBGAUTHSTATUS, Debug Authentication Status register

The DBGAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

Configuration

AArch32 System register DBGAUTHSTATUS bits [31:0] are architecturally mapped to AArch64 System register [DBGAUTHSTATUS_EL1\[31:0\]](#).

AArch32 System register DBGAUTHSTATUS bits [31:0] are architecturally mapped to External register [DBGAUTHSTATUS_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGAUTHSTATUS are UNDEFINED.

This register is required in all implementations.

Attributes

DBGAUTHSTATUS is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|------|-----|-------|------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | SNID | SID | NSNID | NSID | | | | |

Bits [31:8]

Reserved, RES0.

SNID, bits [7:6]

When FEAT_Debugv8p4 is implemented:

Secure Non-Invasive Debug.

This field has the same value as DBGAUTHSTATUS.SID.

Otherwise:

Secure Non-Invasive Debug.

| SNID | Meaning |
|------|---|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR.NS is 1. |
| 0b10 | Implemented and disabled. ExternalSecureNoninvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalSecureNoninvasiveDebugEnabled() == TRUE. |

All other values are reserved.

SID, bits [5:4]

Secure Invasive Debug.

| SID | Meaning |
|------|--|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR_EL3 .NS is 1. |
| 0b10 | Implemented and disabled. ExternalSecureInvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalSecureInvasiveDebugEnabled() == TRUE. |

All other values are reserved.

NSNID, bits [3:2]

When FEAT_Debugv8p4 is implemented:

Non-secure Non-invasive debug.

| NSNID | Meaning |
|-------|---|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR .NS is 0. |
| 0b11 | Implemented and enabled. EL3 is implemented or the Effective value of SCR .NS is 1. |

All other values are reserved.

Otherwise:

Non-secure Non-Invasive Debug.

| NSNID | Meaning |
|-------|---|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR .NS is 0 |
| 0b10 | Implemented and disabled. ExternalNoninvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalNoninvasiveDebugEnabled() == TRUE. |

All other values are reserved.

NSID, bits [1:0]

Non-secure Invasive Debug.

| NSID | Meaning |
|------|---|
| 0b00 | Not implemented. EL3 is not implemented or the Effective value of SCR_EL3 .NS is 0. |
| 0b10 | Implemented and disabled. ExternalInvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalInvasiveDebugEnabled() == TRUE. |

All other values are reserved.

Accessing DBGAUTHSTATUS

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0111 | 0b1110 | 0b110 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGAUTHSTATUS;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            else
                return DBGAUTHSTATUS;
    elsif PSTATE.EL == EL3 then
        return DBGAUTHSTATUS;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGBCR<n>, Debug Breakpoint Control Registers, n = 0 - 15

The DBGBCR<n> characteristics are:

Purpose

Holds control information for a breakpoint. Forms breakpoint n together with value register [DBGBVR<n>](#). If EL2 is implemented and this breakpoint supports Context matching, [DBGBVR<n>](#) can be associated with a Breakpoint Extended Value Register [DBGBXVR<n>](#) for VMID matching.

Configuration

AArch32 System register DBGBCR<n> bits [31:0] are architecturally mapped to AArch64 System register [DBGBCR<n>_EL1\[31:0\]](#).

AArch32 System register DBGBCR<n> bits [31:0] are architecturally mapped to External register [DBGBCR<n>_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGBCR<n> are UNDEFINED.

If breakpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGBCR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----|----|-----|------|----|----|---|-----|---|---|------|-----|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | BT | | | | LBN | | | | SSC | | HMC | RES0 | | | | BAS | | | RES0 | PMC | E | | | |

When the E field is zero, all the other fields in the register are ignored.

Bits [31:24]

Reserved, RES0.

BT, bits [23:20]

Breakpoint Type. Possible values are:

| BT | Meaning |
|--------|---|
| 0b0000 | Unlinked instruction address match. DBGBVR<n> is the address of an instruction. |
| 0b0001 | As 0b0000 with linking enabled. |
| 0b0010 | Unlinked Context ID match. When FEAT_VHE is implemented, EL2 is using AArch64, and the Effective value of HCR_EL2.E2H is 1, if either the PE is executing at EL0 with HCR_EL2.TGE set to 1 or the PE is executing at EL2, then DBGBVR<n>.ContextID must match the CONTEXTIDR_EL2 value. Otherwise, DBGBVR<n>.ContextID must match the CONTEXTIDR value. |
| 0b0011 | As 0b0010 with linking enabled. |
| 0b0100 | Unlinked instruction address mismatch. DBGBVR<n> is the address of an instruction to be stepped. |
| 0b0101 | As 0b0100 with linking enabled. |
| 0b0110 | Unlinked CONTEXTIDR_EL1 match. DBGBVR<n>.ContextID is a Context ID compared against CONTEXTIDR . |
| 0b0111 | As 0b0110 with linking enabled. |
| 0b1000 | Unlinked VMID match. DBGBXVR<n>.VMID is a VMID compared against VTTBR.VMID . |
| 0b1001 | As 0b1000 with linking enabled. |
| 0b1010 | Unlinked VMID and Context ID match. DBGBVR<n>.ContextID is a Context ID compared against CONTEXTIDR , and DBGBXVR<n>.VMID is a VMID compared against VTTBR.VMID . |
| 0b1011 | As 0b1010 with linking enabled. |
| 0b1100 | Unlinked CONTEXTIDR_EL2 match. DBGBXVR<n>.ContextID2 is a Context ID compared against CONTEXTIDR_EL2 . |
| 0b1101 | As 0b1100 with linking enabled. |
| 0b1110 | Unlinked Full Context ID match. DBGBVR<n>.ContextID is compared against CONTEXTIDR , and DBGBXVR<n>.ContextID2 is compared against CONTEXTIDR_EL2 . |
| 0b1111 | As 0b1110 with linking enabled. |

For more information on Breakpoints and their constraints, see 'Breakpoint exceptions' and 'Reserved DBGBCR<n>.BT values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked breakpoint number. For Linked address matching breakpoints, this specifies the index of the Context-matching breakpoint linked to.

For all other breakpoint types this field is ignored and reads of the register return an UNKNOWN value.

This field is ignored when the value of DBGBCR<n>.E is 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the HMC and PMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields.

For more information, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' and 'Reserved DBGBCR<n>.{SSC, HMC, PMC} values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and PMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see the SSC, bits [15:14] description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [12:9]

Reserved, RES0.

BAS, bits [8:5]

Byte address select. Defines which half-words an address-matching breakpoint matches, regardless of the instruction set and Execution state.

The permitted values depend on the breakpoint type.

For Address match breakpoints, the permitted values are:

| BAS | Match instruction at | Constraint for debuggers |
|--------|------------------------------------|--------------------------|
| 0b0011 | DBGBVR<n> | Use for T32 instructions |
| 0b1100 | DBGBVR<n> +2 | Use for T32 instructions |
| 0b1111 | DBGBVR<n> | Use for A32 instructions |

All other values are reserved. For more information, see 'Reserved DBGBCR<n>.BAS values'.

For more information on using the BAS field in Address Match breakpoints, see 'Using the BAS field in Address Match breakpoints'.

For Address mismatch breakpoints in an AArch32 stage 1 translation regime, the permitted values are:

| BAS | Step instruction at | Constraint for debuggers |
|--------|------------------------------------|-------------------------------------|
| 0b0000 | - | Use for a match anywhere breakpoint |
| 0b0011 | DBGBVR<n> | Use for T32 instructions |
| 0b1100 | DBGBVR<n> +2 | Use for T32 instructions |
| 0b1111 | DBGBVR<n> | Use for A32 instructions |

All other values are reserved. For more information, see 'Reserved DBGBCR<n>.BAS values'.

For more information on using the BAS field in address mismatch breakpoints, see 'Using the BAS field in Address Match breakpoints'.

For Context matching breakpoints, this field is RES1 and ignored.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [4:3]

Reserved, RES0.

PMC, bits [2:1]

Privilege mode control. Determines the Exception level or levels at which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and HMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see the DBGBCR<n>.SSC description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable breakpoint [DBGBVR<n>](#). Possible values are:

| E | Meaning |
|-----|----------------------|
| 0b0 | Breakpoint disabled. |
| 0b1 | Breakpoint enabled. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBCR<n>

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | n[3:0] | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGBCR[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGBCR[UInt(CRm<3:0>)];
elsif PSTATE.EL == EL3 then
    if DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGBCR[UInt(CRm<3:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | n[3:0] | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            DBGBCR[UInt(CRm<3:0>)] = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
                Halt(DebugHalt_SoftwareAccess);
            else
                DBGBCR[UInt(CRm<3:0>)] = R[t];
    elsif PSTATE.EL == EL3 then
        if DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            DBGBCR[UInt(CRm<3:0>)] = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

**DBGBVR<n>, Debug Breakpoint Value Registers, n = 0
- 15**

The DBGBVR<n> characteristics are:

Purpose

Holds a value for use in breakpoint matching, either the virtual address of an instruction or a context ID. Forms breakpoint `n` together with control register [DBGBCR<n>](#). If EL2 is implemented and this breakpoint supports Context matching, [DBGBVR<n>](#) can be associated with a Breakpoint Extended Value Register [DBGBXVR<n>](#) for VMID matching.

Configuration

AArch32 System register DBGBVR<n> bits [31:0] are architecturally mapped to AArch64 System register DBGBVR<n> [EL1\[31:0\]](#).

AArch32 System register DBGBVR<n> bits [31:0] are architecturally mapped to External register [DBGBVR<n>_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGVBR<n> are UNDEFINED.

How this register is interpreted depends on the value of [DBGBCR<n>.BT](#).

- When [DBGBCR<n>](#).BT is 0b0x0x, this register holds a virtual address.
- When [DBGBCR<n>](#).BT is 0bxx1x, this register holds a Context ID.

For other values of [DBGBCR<n>.BT](#), this register is RES0.

Some breakpoints might not support Context ID comparison. For more information, see the description of the [DBGDIDR.CTX CMPs](#) field.

If breakpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGBVR<n> is a 32-bit register.

Field descriptions

When DBGBCR<n>.BT == 0b0x0x:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA[31:2] | | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | |

VA[31:2], bits [31:2]

Bits[31:2] of the address value for comparison.

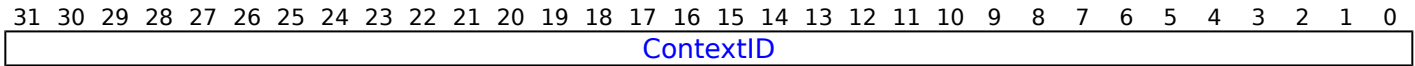
The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When DBGBCR<n>.BT == 0b001x:



ContextID, bits [31:0]

Context ID value for comparison.

The value is compared against [CONTEXTIDR_EL2](#) when all of the following are true:

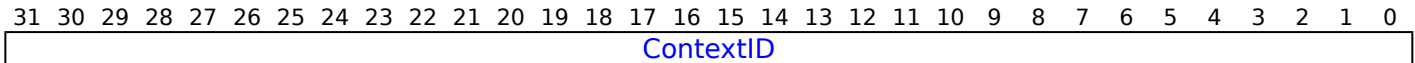
- FEAT_VHE is implemented or FEAT_Debugv8p2 is implemented.
- [HCR_EL2](#).{E2H, TGE} is {1,1}.
- The PE is executing at EL0.
- EL2 is using AArch64 and is enabled in the current Security state.

Otherwise, the value is compared against [CONTEXTIDR](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>.BT == 0b101x and EL2 is implemented:



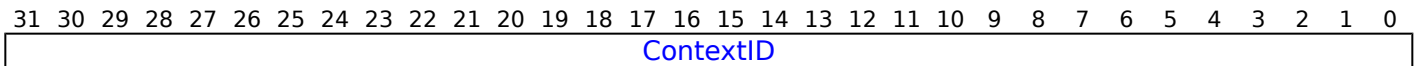
ContextID, bits [31:0]

Context ID value for comparison against [CONTEXTIDR](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>.BT == 0bx11x, EL2 is implemented and (FEAT_VHE is implemented or FEAT_Debugv8p2 is implemented):



ContextID, bits [31:0]

Context ID value for comparison against [CONTEXTIDR](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBVR<n>

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | n[3:0] | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGGBVR[UInt(CRm<3:0>)];
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGGBVR[UInt(CRm<3:0>)];
elseif PSTATE.EL == EL3 then
    if DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        return DBGGBVR[UInt(CRm<3:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | n[3:0] | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGGBVR[UInt(CRm<3:0>)] = R[t];
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGGBVR[UInt(CRm<3:0>)] = R[t];
elseif PSTATE.EL == EL3 then
    if DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGGBVR[UInt(CRm<3:0>)] = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGBXVR<n>, Debug Breakpoint Extended Value Registers, n = 0 - 15

The DBGBXVR<n> characteristics are:

Purpose

Holds a value for use in breakpoint matching, to support VMID matching. Used in conjunction with a control register [DBGBCR<n>](#) and a value register [DBGBVR<n>](#), where EL2 is implemented and breakpoint n supports Context matching.

Configuration

AArch32 System register DBGBXVR<n> bits [31:0] are architecturally mapped to AArch64 System register [DBGBVR<n>_EL1\[63:32\]](#).

AArch32 System register DBGBXVR<n> bits [31:0] are architecturally mapped to External register [DBGBVR<n>_EL1\[63:32\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGBXVR<n> are UNDEFINED.

How this register is interpreted depends on the value of [DBGBCR<n>_BT](#).

- When [DBGBCR<n>_BT](#) is 0b10xx, this register holds a VMID.
- When [DBGBCR<n>_BT](#) is 0b11xx, this register holds a Context ID.

For other values of [DBGBCR<n>_BT](#), this register is RES0.

Accesses to this register are UNDEFINED in any of the following cases:

- Breakpoint n is not implemented.
- Breakpoint n does not support Context matching.
- EL2 is not implemented.

For more information, see the description of the [DBGDIDR.CTX_CMPs](#) field.

Attributes

DBGBXVR<n> is a 32-bit register.

Field descriptions

When DBGBCR<n>_BT == 0b10xx and EL2 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|---|---|-----------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | VMID[15:8] | | | | | | | | VMID[7:0] | | | | | | | |

Bits [31:16]

Reserved, RES0.

VMID[15:8], bits [15:8]

When FEAT_VMID16 is implemented and VTCR_EL2.VS == 1:

Extension to VMID[7:0]. For more information, see VMID[7:0].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID[7:0], bits [7:0]

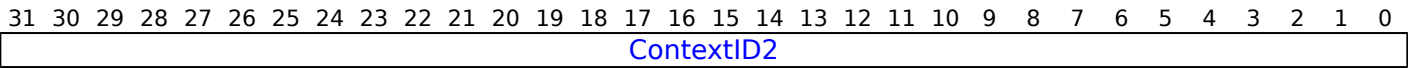
VMID value for comparison. The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- [VTCR_EL2](#).VS is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>.BT == 0b11xx and EL2 is implemented:



ContextID2, bits [31:0]

When FEAT_VHE is implemented or FEAT_Debugv8p2 is implemented:

Context ID value for comparison against [CONTEXTIDR_EL2](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing DBGBXVR<n>

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0001 | n[3:0] | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        elsif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            return DBG BXVR[UInt(CRm<3:0>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            elsif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
                Halt(DebugHalt_SoftwareAccess);
            else
                return DBG BXVR[UInt(CRm<3:0>)];
    elsif PSTATE.EL == EL3 then
        if DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            return DBG BXVR[UInt(CRm<3:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0001 | n[3:0] | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        elsif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            DBG BXVR[UInt(CRm<3:0>)] = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            elsif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
                Halt(DebugHalt_SoftwareAccess);
            else
                DBG BXVR[UInt(CRm<3:0>)] = R[t];
    elsif PSTATE.EL == EL3 then
        if DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            DBG BXVR[UInt(CRm<3:0>)] = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGCLAIMCLR, Debug CLAIM Tag Clear register

The DBGCLAIMCLR characteristics are:

Purpose

Used by software to read the values of the CLAIM tag bits, and to clear CLAIM tag bits to 0.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the [DBGCLAIMSET](#) register.

Configuration

AArch32 System register DBGCLAIMCLR bits [31:0] are architecturally mapped to AArch64 System register [DBGCLAIMCLR_EL1\[31:0\]](#).

AArch32 System register DBGCLAIMCLR bits [31:0] are architecturally mapped to External register [DBGCLAIMCLR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGCLAIMCLR are UNDEFINED.

An implementation must include eight CLAIM tag bits.

Attributes

DBGCLAIMCLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RAZ/WI | | | | | | | | | | | | | | | | | | | | | | | | CLAIM | | | | | | | |

Bits [31:8]

Reserved, RAZ/WI.

CLAIM, bits [7:0]

Read or clear CLAIM tag bits. Reading this field returns the current value of the CLAIM tag bits.

Writing a 1 to one of these bits clears the corresponding CLAIM tag bit to 0. This is an indirect write to the CLAIM tag bits. A single write operation can clear multiple CLAIM tag bits to 0.

Writing 0 to one of these bits has no effect.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Accessing DBGCLAIMCLR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0111 | 0b1001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGCLAIMCLR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGCLAIMCLR;
elsif PSTATE.EL == EL3 then
    return DBGCLAIMCLR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0111 | 0b1001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            DBGCLAIMCLR = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            DBGCLAIMCLR = R[t];
elsif PSTATE.EL == EL3 then
    DBGCLAIMCLR = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGCLAIMSET, Debug CLAIM Tag Set register

The DBGCLAIMSET characteristics are:

Purpose

Used by software to set the CLAIM tag bits to 1.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the [DBGCLAIMCLR](#) register.

Configuration

AArch32 System register DBGCLAIMSET bits [31:0] are architecturally mapped to AArch64 System register [DBGCLAIMSET_EL1\[31:0\]](#).

AArch32 System register DBGCLAIMSET bits [31:0] are architecturally mapped to External register [DBGCLAIMSET_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGCLAIMSET are UNDEFINED.

An implementation must include eight CLAIM tag bits.

Attributes

DBGCLAIMSET is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RAZ/WI | | | | | | | | | | | | | | | | | | | | | | | | CLAIM | | | | | | | |

Bits [31:8]

Reserved, RAZ/WI.

CLAIM, bits [7:0]

Set CLAIM tag bits.

This field is RAO.

Writing a 1 to one of these bits sets the corresponding CLAIM tag bit to 1. This is an indirect write to the CLAIM tag bits. A single write operation can set multiple CLAIM tag bits to 1.

Writing 0 to one of these bits has no effect.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Accessing DBGCLAIMSET

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0111 | 0b1000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGCLAIMSET;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            else
                return DBGCLAIMSET;
    elsif PSTATE.EL == EL3 then
        return DBGCLAIMSET;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0111 | 0b1000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            DBGCLAIMSET = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            DBGCLAIMSET = R[t];
elsif PSTATE.EL == EL3 then
    DBGCLAIMSET = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDCCINT, DCC Interrupt Enable Register

The DBGDCCINT characteristics are:

Purpose

Enables interrupt requests to be signaled based on the DCC status flags.

Configuration

AArch32 System register DBGDCCINT bits [31:0] are architecturally mapped to AArch64 System register [MDCCINT_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGDCCINT are UNDEFINED.

Attributes

DBGDCCINT is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|------|---|---|---|--|--|--|--|--|--|--|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | |
| RES0 | RX | TX | | | | | | | | | | | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | |

Bit [31]

Reserved, RES0.

RX, bit [30]

DCC interrupt request enable control for DTRRX. Enables a common **COMMIRQ** interrupt request to be signaled based on the DCC status flags.

| RX | Meaning |
|-----|---|
| 0b0 | No interrupt request generated by DTRRX. |
| 0b1 | Interrupt request will be generated on RXfull == 1. |

If legacy **COMMRX** and **COMMTX** signals are implemented, then these are not affected by the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

TX, bit [29]

DCC interrupt request enable control for DTRTX. Enables a common **COMMIRQ** interrupt request to be signaled based on the DCC status flags.

| TX | Meaning |
|-----|---|
| 0b0 | No interrupt request generated by DTRTX. |
| 0b1 | Interrupt request will be generated on TXfull == 0. |

If legacy **COMMRX** and **COMMTX** signals are implemented, then these are not affected by the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [28:0]

Reserved, RES0.

Accessing DBGDCCINT

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    return DBGDCCINT;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDCCINT;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDCCINT;
elseif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else

```

```
return DBGDCCINT;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    DBGDCCINT = R[t];
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDCCINT = R[t];
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDCCINT = R[t];
elseif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else

```

```
DBGDCCINT = R[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDEVID, Debug Device ID register 0

The DBGDEVID characteristics are:

Purpose

Adds to the information given by the [DBGDIDR](#) by describing other features of the debug implementation.

Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGDEVID are UNDEFINED.

This register is required in all implementations.

Attributes

DBGDEVID is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------|----|----|----|-------------------------|----|----|----|----------------------------|----|----|----|---------------------------|----|----|----|-----------------------------|----|----|----|----------------------------|----|---|---|----------------------------|---|---|---|--------------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CIDMask | | | | AuxRegs | | | | DoubleLock | | | | VirtExtns | | | | VectorCatch | | | | BPAddrMask | | | | WPAddrMask | | | | PCSample | | | |

CIDMask, bits [31:28]

Indicates the level of support for the Context ID matching breakpoint masking capability. Defined values are:

| CIDMask | Meaning |
|---------|--|
| 0b0000 | Context ID masking is not implemented. |
| 0b0001 | Context ID masking is implemented. |

All other values are reserved. The value of this for Armv8 is 0b0000.

AuxRegs, bits [27:24]

Indicates support for Auxiliary registers. Permitted values for this field are:

| AuxRegs | Meaning |
|---------|--|
| 0b0000 | None supported. |
| 0b0001 | Support for External Debug Auxiliary Control Register, EDACR . |

All other values are reserved.

DoubleLock, bits [23:20]

OS Double Lock implemented. Defined values are:

| DoubleLock | Meaning |
|------------|--|
| 0b0000 | OS Double Lock is not implemented. DBGOSDLR is RAZ/WI. |
| 0b0001 | OS Double Lock is implemented. DBGOSDLR is RW. |

FEAT_DoubleLock implements the functionality identified by the value 0b0001.

All other values are reserved.

VirtExtns, bits [19:16]

Indicates whether EL2 is implemented. Defined values are:

| VirtExtns | Meaning |
|------------------|-------------------------|
| 0b0000 | EL2 is not implemented. |
| 0b0001 | EL2 is implemented. |

All other values are reserved.

VectorCatch, bits [15:12]

Defines the form of Vector Catch exception implemented. Defined values are:

| VectorCatch | Meaning |
|--------------------|--|
| 0b0000 | Address matching Vector Catch exception implemented. |
| 0b0001 | Exception matching Vector Catch exception implemented. |

All other values are reserved.

BPAAddrMask, bits [11:8]

Indicates the level of support for the instruction address matching breakpoint masking capability. Defined values are:

| BPAAddrMask | Meaning |
|--------------------|---|
| 0b0000 | Breakpoint address masking might be implemented. If not implemented, DBGBCR<n> [28:24] is RAZ/WI. |
| 0b0001 | Breakpoint address masking is implemented. |
| 0b1111 | Breakpoint address masking is not implemented. DBGBCR<n> [28:24] is RES0. |

All other values are reserved. The value of this for Armv8 is 0b1111.

WPAAddrMask, bits [7:4]

Indicates the level of support for the data address matching watchpoint masking capability. Defined values are:

| WPAAddrMask | Meaning |
|--------------------|--|
| 0b0000 | Watchpoint address masking might be implemented. If not implemented, DBGWCR<n> .MASK (Address mask) is RAZ/WI. |
| 0b0001 | Watchpoint address masking is implemented. |
| 0b1111 | Watchpoint address masking is not implemented. DBGWCR<n> .MASK (Address mask) is RES0. |

All other values are reserved. The value of this for Armv8 is 0b0001.

PCSample, bits [3:0]

Indicates the level of PC Sample-based Profiling support using external debug registers. Defined values are:

| PCSample | Meaning |
|-----------------|---|
| 0b0000 | PC Sample-based Profiling Extension is not implemented in the external debug registers space. |
| 0b0010 | Only EDPCSR and EDCIDS are implemented. This option is only permitted if EL3 and EL2 are not implemented. |
| 0b0011 | EDPCSR , EDCIDS , and EDVIDSR are implemented. |

All other values are reserved.

When FEAT_PCSRv8p2 is implemented, the only permitted value is 0b0000.

Note

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of [PMDEVID.PCSample](#).

Accessing DBGDEVID

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0111 | 0b0010 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGDEVID;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            else
                return DBGDEVID;
    elsif PSTATE.EL == EL3 then
        return DBGDEVID;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDEVID1, Debug Device ID register 1

The DBGDEVID1 characteristics are:

Purpose

Adds to the information given by the [DBGDIDR](#) by describing other features of the debug implementation.

Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGDEVID1 are UNDEFINED.

This register is required in all implementations.

Attributes

DBGDEVID1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | PCSROffset | | | |

Bits [31:4]

Reserved, RES0.

PCSROffset, bits [3:0]

This field indicates the offset applied to PC samples returned by reads of [EDPCSR](#). Permitted values of this field in Armv8 are:

| PCSROffset | Meaning |
|------------|--|
| 0b0000 | EDPCSR is not implemented. |
| 0b0010 | EDPCSR implemented. Samples have no offset applied and do not sample the instruction set state in AArch32 state. |

When FEAT_PCSRv8p2 is implemented, the only permitted value is 0b0000.

Note

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of [PMDEVID](#).PCSample.

Accessing DBGDEVID1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0111 | 0b0001 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGDEVID1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            else
                return DBGDEVID1;
    elsif PSTATE.EL == EL3 then
        return DBGDEVID1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDEVID2, Debug Device ID register 2

The DBGDEVID2 characteristics are:

Purpose

Reserved for future descriptions of features of the debug implementation.

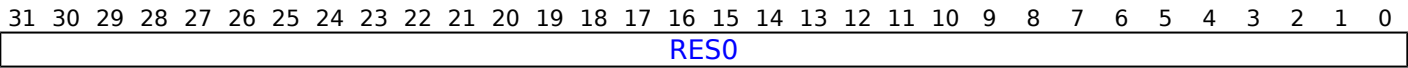
Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGDEVID2 are UNDEFINED.

Attributes

DBGDEVID2 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing DBGDEVID2

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0111 | 0b0000 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGDEVID2;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGDEVID2;
    elsif PSTATE.EL == EL3 then
        return DBGDEVID2;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDIDR, Debug ID Register

The DBGDIDR characteristics are:

Purpose

Specifies which version of the Debug architecture is implemented, and some features of the debug implementation.

Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGDIDR are UNDEFINED.

If EL1 cannot use AArch32 then the implementation of this register is OPTIONAL and deprecated.

Attributes

DBGDIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|------|----|----|----|----------|----|----|----|---------|----|----|----|------|-----------|------|--------|------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| WRPs | | | | BRPs | | | | CTX_CMPs | | | | Version | | | | RES1 | nSUHD_imp | RES0 | SE_imp | RES0 | | | | | | | | | | | |

WRPs, bits [31:28]

The number of watchpoints implemented, minus 1.

Permitted values of this field are from 0b0001 for 2 implemented watchpoints, to 0b1111 for 16 implemented watchpoints.

The value of 0b0000 is reserved.

If AArch64 is implemented, this field has the same value as [ID_AA64DFR0_EL1](#).WRPs.

BRPs, bits [27:24]

The number of breakpoints implemented, minus 1.

Permitted values of this field are from 0b0001 for 2 implemented breakpoint, to 0b1111 for 16 implemented breakpoints.

The value of 0b0000 is reserved.

If AArch64 is implemented, this field has the same value as [ID_AA64DFR0_EL1](#).BRPs.

CTX_CMPs, bits [23:20]

The number of breakpoints that can be used for Context matching, minus 1.

Permitted values of this field are from 0b0000 for 1 Context matching breakpoint, to 0b1111 for 16 Context matching breakpoints.

The Context matching breakpoints must be the highest addressed breakpoints. For example, if six breakpoints are implemented and two are Context matching breakpoints, they must be breakpoints 4 and 5.

If AArch64 is implemented, this field has the same value as [ID_AA64DFR0_EL1](#).CTX_CMPs.

Version, bits [19:16]

Debug architecture version. Indicates presence of Armv8 debug architecture. Defined values are:

| Version | Meaning |
|----------------|--|
| 0b0000 | Not supported. |
| 0b0001 | Armv6, v6 Debug architecture, with System registers access. |
| 0b0010 | Armv6, v6.1 Debug architecture, with System registers access. |
| 0b0011 | Armv7, v7 Debug architecture, with only baseline System registers. |
| 0b0100 | Armv7, v7 Debug architecture, with all System registers implemented. |
| 0b0101 | Armv7, v7.1 Debug architecture, with System registers access. |
| 0b0110 | Armv8 debug architecture. |
| 0b0111 | Armv8 debug architecture with Virtualization Host Extensions. |
| 0b1000 | Armv8.2 debug architecture, FEAT_Debugv8p2. |
| 0b1001 | Armv8.4 debug architecture, FEAT_Debugv8p4. |
| 0b1010 | Armv8.8 debug architecture, FEAT_Debugv8p8. |

All other values are reserved.

The values 0b0000, 0b0001, 0b0010, 0b0011, 0b0100, and 0b0101 are not permitted in Armv8.

FEAT_VHE adds the functionality identified by the value 0b0111.

FEAT_Debugv8p2 adds the functionality identified by the value 0b1000.

FEAT_Debugv8p4 adds the functionality identified by the value 0b1001.

FEAT_Debugv8p8 adds the functionality identified by the value 0b1010.

From Armv8.1, when FEAT_VHE is implemented the value 0b0110 is not permitted.

From Armv8.2, the values 0b0110 and 0b0111 are not permitted.

From Armv8.4, the value 0b1000 is not permitted.

From Armv8.8, the value 0b1001 is not permitted.

Bit [15]

Reserved, RES1.

nSUHD_imp, bit [14]

In Armv7-A, was Secure User Halting Debug not implemented.

The value of this bit must match the value of the SE_imp bit.

Bit [13]

Reserved, RES0.

SE_imp, bit [12]

EL3 implemented. The meanings of the values of this bit are:

| SE_imp | Meaning |
|---------------|----------------------|
| 0b0 | EL3 not implemented. |
| 0b1 | EL3 implemented. |

The value of this bit must match the value of the nSUHD_imp bit.

Bits [11:0]

Reserved, RES0.

Accessing DBGDIDR

Arm deprecates any access to this register from EL0.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0000 | 0b000 |

```

if Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    return DBGDIDR;
elsif PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
        elsif ELUsingAArch32(EL1) && DBGDSCRext.UCCdis == '1' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> !=
'00') then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDA> != '00') then
                AArch32.TakeHypTrapException(0x05);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
                else
                    return DBGDIDR;
            elsif PSTATE.EL == EL1 then
                if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    UNDEFINED;
                elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
                    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
                elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
                    AArch32.TakeHypTrapException(0x05);
                elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
                    else
                        return DBGDIDR;
            elsif PSTATE.EL == EL2 then
                if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    UNDEFINED;
                elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
                    else
                        return DBGDIDR;
            elsif PSTATE.EL == EL3 then
                return DBGDIDR;

```

DBGDRAR, Debug ROM Address Register

The DBGDRAR characteristics are:

Purpose

Defines the base physical address of a 4KB-aligned memory-mapped debug component, usually a ROM table that locates and describes the memory-mapped debug components in the system. Armv8 deprecates any use of this register.

Configuration

AArch32 System register DBGDRAR bits [63:0] are architecturally mapped to AArch64 System register [MDRAR_EL1\[63:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGDRAR are UNDEFINED.

DBGDRAR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, bits [31:0] are read.

If EL1 cannot use AArch32 then the implementation of this register is OPTIONAL and deprecated.

Attributes

DBGDRAR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | ROMADDR[47:12] | | | | | | | | | | | | | | | |
| ROMADDR[47:12] | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | Valid | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

ROMADDR[47:12], bits [47:12]

Bits[47:12] of the ROM table physical address.

If the physical address size in bits (PAsize) is less than 48 then the register bits corresponding to ROMADDR [47:PAsize] are RES0.

Bits [11:0] of the ROM table physical address are zero.

Arm strongly recommends that bits ROMADDR[(PAsize-1):32] are zero in any system that supports AArch32 at the highest implemented Exception level.

In an implementation that includes EL3, ROMADDR is an address in Non-secure memory. It is IMPLEMENTATION DEFINED whether the ROM table is also accessible in Secure memory.

If DBGDRAR.Valid == 0b00, then this field is UNKNOWN.

Bits [11:2]

Reserved, RES0.

Valid, bits [1:0]

This field indicates whether the ROM Table address is valid.

| Valid | Meaning |
|-------|---|
| 0b00 | ROM Table address is not valid. Software must ignore ROMADDR. |
| 0b11 | ROM Table address is valid. |

Other values are reserved.

Arm recommends implementations set this field to zero.

Accessing DBGDRAR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0001 | 0b0000 | 0b000 |

```

if Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    return DBGDRAR<31:0>;
elsif PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
        elsif ELUsingAArch32(EL1) && DBGDSCRExt.UDCCdis == '1' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDRA> !=
'00') then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDRA> != '00') then
                AArch32.TakeHypTrapException(0x05);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
                else
                    return DBGDRAR<31:0>;
            elsif PSTATE.EL == EL1 then
                if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    UNDEFINED;
                elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDRA> != '00' then
                    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
                elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDRA> != '00' then
                    AArch32.TakeHypTrapException(0x05);
                elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
                    else
                        return DBGDRAR<31:0>;
            elsif PSTATE.EL == EL2 then
                if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    UNDEFINED;
                elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
                    else
                        return DBGDRAR<31:0>;
            elsif PSTATE.EL == EL3 then
                return DBGDRAR<31:0>;

```

MRRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1110 | 0b0001 | 0b0000 |

```

if Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    return DBGDRAR;
elsif PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x0C);
        elsif ELUsingAArch32(EL1) && DBGDSCRext.UCCdis == '1' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDRA> !=
'00') then
                AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDRA> != '00') then
                AArch32.TakeHypTrapException(0x0C);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x0C);
                else
                    return DBGDRAR;
            elsif PSTATE.EL == EL1 then
                if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    UNDEFINED;
                elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDRA> != '00' then
                    AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
                elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDRA> != '00' then
                    AArch32.TakeHypTrapException(0x0C);
                elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.AArch32SystemAccessTrap(EL3, 0x0C);
                    else
                        return DBGDRAR;
            elsif PSTATE.EL == EL2 then
                if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    UNDEFINED;
                elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.AArch32SystemAccessTrap(EL3, 0x0C);
                    else
                        return DBGDRAR;
            elsif PSTATE.EL == EL3 then
                return DBGDRAR;

```

DBGDSAR, Debug Self Address Register

The DBGDSAR characteristics are:

Purpose

In earlier versions of the Arm Architecture, this register defines the offset from the base address defined in [DBGDRAR](#) of the physical base address of the debug registers for the PE. Armv8 deprecates any use of this register.

Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGDSAR are UNDEFINED.

DBGDSAR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, bits [31:0] are read.

If EL1 cannot use AArch32 then the implementation of this register is OPTIONAL and deprecated.

Attributes

DBGDSAR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | RAZ |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:2]

Reserved, RES0.

Bits [1:0]

Reserved, RAZ.

This field indicates whether the debug self address offset is valid. For ARMv8, this field is always 0b00, the offset is not valid.

Accessing DBGDSAR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0010 | 0b0000 | 0b000 |

```

if Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    return DBGDSAR<31:0>;
elsif PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
        elsif ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDRA> !=
'00') then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDRA> != '00') then
                AArch32.TakeHypTrapException(0x05);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x05);
                else
                    return DBGDSAR<31:0>;
            elsif PSTATE.EL == EL1 then
                if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    UNDEFINED;
                elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDRA> != '00' then
                    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
                elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDRA> != '00' then
                    AArch32.TakeHypTrapException(0x05);
                elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
                    else
                        return DBGDSAR<31:0>;
            elsif PSTATE.EL == EL2 then
                if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    UNDEFINED;
                elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
                    else
                        return DBGDSAR<31:0>;
            elsif PSTATE.EL == EL3 then
                return DBGDSAR<31:0>;

```

MRRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1110 | 0b0010 | 0b0000 |

```

if Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    return DBGDSAR;
elsif PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x0C);
        elsif ELUsingAArch32(EL1) && DBGDSCRext.UCCdis == '1' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDRA> !=
'00') then
                AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDRA> != '00') then
                AArch32.TakeHypTrapException(0x0C);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x0C);
                else
                    return DBGDSAR;
            elsif PSTATE.EL == EL1 then
                if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    UNDEFINED;
                elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDRA> != '00' then
                    AArch64.AArch32SystemAccessTrap(EL2, 0x0C);
                elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDRA> != '00' then
                    AArch32.TakeHypTrapException(0x0C);
                elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.AArch32SystemAccessTrap(EL3, 0x0C);
                    else
                        return DBGDSAR;
                elsif PSTATE.EL == EL2 then
                    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                        UNDEFINED;
                    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                        if Halted() && EDSCR.SDD == '1' then
                            UNDEFINED;
                        else
                            AArch64.AArch32SystemAccessTrap(EL3, 0x0C);
                        else
                            return DBGDSAR;
                    elsif PSTATE.EL == EL3 then
                        return DBGDSAR;

```

DBGDSCRext, Debug Status and Control Register, External View

The DBGDSCRext characteristics are:

Purpose

Main control register for the debug implementation.

Configuration

AArch32 System register DBGDSCRext bits [31:0] are architecturally mapped to AArch64 System register [MDSCR_EL1\[31:0\]](#).

AArch32 System register DBGDSCRext bit [15] is architecturally mapped to AArch32 System register [DBGDSCRint\[15\]](#).

AArch32 System register DBGDSCRext bit [12] is architecturally mapped to AArch32 System register [DBGDSCRint\[12\]](#).

AArch32 System register DBGDSCRext bits [5:2] are architecturally mapped to AArch32 System register [DBGDSCRint\[5:2\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGDSCRext are UNDEFINED.

This register is required in all implementations.

Attributes

DBGDSCRext is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|------------------------|------------------------|----------------------|------------------------|----------------------|------------------------|---------------------|----------------------|-----------------------|--------------------------|-------------------------|------------------------|---------------------|----------------------|-------------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 |
| TFO | RXfull | TXfull | RES0 | RXOTXU | RES0 | INTdis | TDA | RES0 | SC2NS | SPNIDdis | SPIDdis | MDBGen | HDE | RES0 | UDCCdis | RES0 | ERRM | ERRM | ERRM | ERRM | ERRM | ERRM | ERRM | ERRM | ERRM | ERRM |

TFO, bit [31]

When FEAT_TRF is implemented:

Trace Filter override. Used for save/restore of [EDSCR.TFO](#).

When the OS Lock is unlocked, [DBGOSLSR.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When the OS Lock is locked, [DBGOSLSR.OSLK](#) == 1, this bit holds the value of [EDSCR.TFO](#). Reads and writes of this bit are indirect accesses to [EDSCR.TFO](#).

Accessing this field has the following behavior:

- When [DBGOSLSR.OSLK](#) == 1, access to this field is **RW**.
- When [DBGOSLSR.OSLK](#) == 0, access to this field is **RO**.

Otherwise:

Reserved, RES0.

RXfull, bit [30]

DTRRX full. Used for save/restore of [EDSCR.RXfull](#).

When [DBGOSLSR.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [DBGOSLSR.OSLK](#) == 1, this bit holds the value of [EDSCR.RXfull](#). Reads and writes of this bit are indirect accesses to [EDSCR.RXfull](#).

Arm deprecates use of this bit other than for save/restore. Use [DBGDSCRint](#) to access the DTRRX full status.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When [DBGOSLSR.OSLK](#) == 1, access to this field is **RW**.
- When [DBGOSLSR.OSLK](#) == 0, access to this field is **RO**.

TXfull, bit [29]

DTRTX full. Used for save/restore of [EDSCR.TXfull](#).

When [DBGOSLSR.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [DBGOSLSR.OSLK](#) == 1, this bit holds the value of [EDSCR.TXfull](#). Reads and writes of this bit are indirect accesses to [EDSCR.TXfull](#).

Arm deprecates use of this bit other than for save/restore. Use [DBGDSCRint](#) to access the DTRTX full status.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When [DBGOSLSR.OSLK](#) == 1, access to this field is **RW**.
- When [DBGOSLSR.OSLK](#) == 0, access to this field is **RO**.

Bit [28]

Reserved, RES0.

RXO, bit [27]

Used for save/restore of [EDSCR.RXO](#).

When [DBGOSLSR.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [DBGOSLSR.OSLK](#) == 1, this bit holds the value of [EDSCR.RXO](#). Reads and writes of this bit are indirect accesses to [EDSCR.RXO](#).

When [DBGOSLSR.OSLK](#) == 1, if bits [27,6] of the value written to [DBGDSCRext](#) are {1,0}, that is, the RXO bit is 1 and the ERR bit is 0, the PE sets [EDSCR.{RXO,ERR}](#) to UNKNOWN values.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When [DBGOSLSR.OSLK](#) == 1, access to this field is **RW**.
- When [DBGOSLSR.OSLK](#) == 0, access to this field is **RO**.

TXU, bit [26]

Used for save/restore of [EDSCR.TXU](#).

When [DBGOSLSR.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [DBGOSLSR.OSLK](#) == 1, this bit holds the value of [EDSCR.TXU](#). Reads and writes of this bit are indirect accesses to [EDSCR.TXU](#).

When [DBGOSLSR.OSLK](#) == 1, if bits [26,6] of the value written to DBGDSCRext are {1,0}, that is, the TXU bit is 1 and the ERR bit is 0, the PE sets [EDSCR.{TXU,ERR}](#) to UNKNOWN values.

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When [DBGOSLSR.OSLK](#) == 1, access to this field is **RW**.
- When [DBGOSLSR.OSLK](#) == 0, access to this field is **RO**.

Bits [25:24]

Reserved, RES0.

INTdis, bits [23:22]

Used for save/restore of [EDSCR.INTdis](#).

When [DBGOSLSR.OSLK](#) == 0, this field is RO, and software must treat it as UNK/SBZP.

When [DBGOSLSR.OSLK](#) == 1, this field is RW and holds the value of [EDSCR.INTdis](#). Reads and writes of this field are indirect accesses to [EDSCR.INTdis](#).

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When [DBGOSLSR.OSLK](#) == 1, access to this field is **RW**.
- When [DBGOSLSR.OSLK](#) == 0, access to this field is **RO**.

TDA, bit [21]

Used for save/restore of [EDSCR.TDA](#).

When [DBGOSLSR.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [DBGOSLSR.OSLK](#) == 1, this bit holds the value of [EDSCR.TDA](#). Reads and writes of this bit are indirect accesses to [EDSCR.TDA](#).

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When [DBGOSLSR.OSLK](#) == 1, access to this field is **RW**.
- When [DBGOSLSR.OSLK](#) == 0, access to this field is **RO**.

Bit [20]

Reserved, RES0.

SC2, bit [19]

When FEAT_PCSRv8 is implemented, FEAT_VHE is implemented and FEAT_PCSRv8p2 is not implemented:

Used for save/restore of [EDSCR.SC2](#).

When [DBGOSLSR.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [DBGOSLSR.OSLK](#) == 1, this bit holds the value of [EDSCR.SC2](#). Reads and writes of this bit are indirect accesses to [EDSCR.SC2](#).

Accessing this field has the following behavior:

- When [DBGOSLSR.OSLK](#) == 1, access to this field is **RW**.
- When [DBGOSLSR.OSLK](#) == 0, access to this field is **RO**.

Otherwise:

Reserved, RES0.

NS, bit [18]

Non-secure status.

Arm deprecates use of this field.

| NS | Meaning |
|-----|-------------------|
| 0b0 | Secure state. |
| 0b1 | Non-secure state. |

Access to this field is **RO**.

SPNIDdis, bit [17]**When EL3 is implemented:**

Secure privileged profiling disabled status bit.

| SPNIDdis | Meaning |
|----------|--|
| 0b0 | Profiling allowed in Secure privileged modes. |
| 0b1 | Profiling prohibited in Secure privileged modes. |

This field reads as 0 if any of the following applies, and reads as 1 otherwise:

- FEAT_Debugv8p2 is not implemented and ExternalSecureNoninvasiveDebugEnabled() returns TRUE.
- EL3 is using AArch32 and the value of [SDCR.SPME](#) is 1.
- EL3 is using AArch64 and the value of [MDCR_EL3.SPME](#) is 1.

Arm deprecates use of this field.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

SPIDdis, bit [16]**When EL3 is implemented:**

Secure privileged AArch32 invasive self-hosted debug disabled status bit. The value of this bit depends on the value of [SDCR.SPD](#) and the pseudocode function `AArch32.SelfHostedSecurePrivilegedInvasiveDebugEnabled()`.

| SPIDdis | Meaning |
|---------|--|
| 0b0 | Self-hosted debug enabled in Secure privileged AArch32 modes. |
| 0b1 | Self-hosted debug disabled in Secure privileged AArch32 modes. |

This bit reads as 1 if any of the following is true and reads as 0 otherwise:

- EL3 is using AArch32 and [SDCR.SPD](#) has the value 0b10.
- EL3 is using AArch64 and [MDCR_EL3.SPD32](#) has the value 0b10.
- EL3 is using AArch32, [SDCR.SPD](#) has the value 0b00, and `AArch32.SelfHostedSecurePrivilegedInvasiveDebugEnabled()` returns FALSE.
- EL3 is using AArch64, [MDCR_EL3.SPD32](#) has the value 0b00, and `AArch32.SelfHostedSecurePrivilegedInvasiveDebugEnabled()` returns FALSE.

Arm deprecates use of this field.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

MDBGGen, bit [15]

Monitor debug events enable. Enable Breakpoint, Watchpoint, and Vector Catch exceptions.

| MDBGGen | Meaning |
|---------|---|
| 0b0 | Breakpoint, Watchpoint, and Vector Catch exceptions disabled. |
| 0b1 | Breakpoint, Watchpoint, and Vector Catch exceptions enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

HDE, bit [14]

Used for save/restore of [EDSCR.HDE](#).

When [DBGOSLSR.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [DBGOSLSR.OSLK](#) == 1, this bit holds the value of [EDSCR.HDE](#). Reads and writes of this bit are indirect accesses to [EDSCR.HDE](#).

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When [DBGOSLSR.OSLK](#) == 1, access to this field is **RW**.
- When [DBGOSLSR.OSLK](#) == 0, access to this field is **RO**.

Bit [13]

Reserved, RES0.

UDCCdis, bit [12]

Traps EL0 accesses to the DCC registers to Undefined mode.

| UDCCdis | Meaning |
|---------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | EL0 accesses to the DBGDSCRint , DBGDTRRXint , DBGDTRTXint , DBGDIDR , DBGDSAR , and DBGDRAR are trapped to Undefined mode. |

Note

All accesses to these registers are trapped, including LDC and STC accesses to [DBGDTRTXint](#) and [DBGDTRRXint](#), and MRRC accesses to [DBGDSAR](#) and [DBGDRAR](#).

Traps of EL0 accesses to the [DBGDTRRXint](#) and [DBGDTRTXint](#) are ignored in Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [11:7]

Reserved, RES0.

ERR, bit [6]

Used for save/restore of [EDSCR.ERR](#).

When [DBGOSLSR.OSLK](#) == 0, software must treat this bit as UNK/SBZP.

When [DBGOSLSR.OSLK](#) == 1, this bit holds the value of [EDSCR.ERR](#). Reads and writes of this bit are indirect accesses to [EDSCR.ERR](#).

The architected behavior of this field determines the value it returns after a reset.

Accessing this field has the following behavior:

- When [DBGOSLSR.OSLK](#) == 1, access to this field is **RW**.
- When [DBGOSLSR.OSLK](#) == 0, access to this field is **RO**.

MOE, bits [5:2]

Method of Entry for debug exception. When a debug exception is taken to an Exception level using AArch32, this field is set to indicate the event that caused the exception:

| MOE | Meaning |
|--------|---|
| 0b0001 | Breakpoint. |
| 0b0011 | Software breakpoint (BKPT) instruction. |
| 0b0101 | Vector catch. |
| 0b1010 | Watchpoint. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

Accessing DBGDSCRext

Individual fields within this register might have restricted accessibility when the OS Lock is unlocked, [DBGOSLSR.OSLK](#) == 0. See the field descriptions for more detail.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGDSCRext;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDSCRext;
elsif PSTATE.EL == EL3 then
    return DBGDSCRext;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDSCRext = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDSCRext = R[t];
elsif PSTATE.EL == EL3 then
    DBGDSCRext = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDSCRint, Debug Status and Control Register, Internal View

The DBGDSCRint characteristics are:

Purpose

Main control register for the debug implementation. This is an internal, read-only view.

Configuration

AArch32 System register DBGDSCRint bits [30:29] are architecturally mapped to AArch64 System register [MDCCSR_EL0\[30:29\]](#).

AArch32 System register DBGDSCRint bit [15] is architecturally mapped to AArch64 System register [MDSCR_EL1\[15\]](#).

AArch32 System register DBGDSCRint bit [12] is architecturally mapped to AArch64 System register [MDSCR_EL1\[12\]](#).

AArch32 System register DBGDSCRint bits [5:2] are architecturally mapped to AArch64 System register [MDSCR_EL1\[5:2\]](#).

AArch32 System register DBGDSCRint bit [15] is architecturally mapped to AArch32 System register [DBGDSCRext\[15\]](#).

AArch32 System register DBGDSCRint bit [12] is architecturally mapped to AArch32 System register [DBGDSCRext\[12\]](#).

AArch32 System register DBGDSCRint bits [5:2] are architecturally mapped to AArch32 System register [DBGDSCRext\[5:2\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGDSCRint are UNDEFINED.

This register is required in all implementations.

DBGDSCRint.{NS, SPNIDdis, SPIDdis, MDBGen, UDCCdis, MOE} are UNKNOWN when the register is accessed at EL0. However, although these values are not accessible at EL0 by instructions that are neither UNPREDICTABLE nor return UNKNOWN values, it is permissible for an implementation to return the values of DBGDSCRext.{NS, SPNIDdis, SPIDdis, MDBGen, UDCCdis, MOE} for these fields at EL0.

It is also permissible for an implementation to return the same values as defined for a read of DBGDSCRint at EL1 or above. (This is the case even if the implementation does not support AArch32 at EL1 or above.)

Attributes

DBGDSCRint is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|--------|--------|------|----|----|----|----|----------|---------|--------|------|---------|------|----|----|-----|------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | RXfull | TXfull | RES0 | | | | NS | SPNIDdis | SPIDdis | MDBGen | RES0 | UDCCdis | RES0 | | | MOE | RES0 | | | | | | | | | | | | | | |

Bit [31]

Reserved, RES0.

RXfull, bit [30]

DTRRX full. Read-only view of the equivalent bit in the [EDSCR](#).

TXfull, bit [29]

DTRTX full. Read-only view of the equivalent bit in the [EDSCR](#).

Bits [28:19]

Reserved, RES0.

NS, bit [18]

Non-secure status.

Read-only view of the equivalent bit in the [DBGDSCRext](#). Arm deprecates use of this field.

SPNIDdis, bit [17]

Secure privileged non-invasive debug disable.

Read-only view of the equivalent bit in the [DBGDSCRext](#). Arm deprecates use of this field.

SPIDdis, bit [16]

Secure privileged invasive debug disable.

Read-only view of the equivalent bit in the [DBGDSCRext](#). Arm deprecates use of this field.

MDBGen, bit [15]

Monitor debug events enable.

Read-only view of the equivalent bit in the [DBGDSCRext](#).

Bits [14:13]

Reserved, RES0.

UDCCdis, bit [12]

User mode access to Debug Communications Channel disable.

Read-only view of the equivalent bit in the [DBGDSCRext](#). Arm deprecates use of this field.

Bits [11:6]

Reserved, RES0.

MOE, bits [5:2]

Method of Entry for debug exception. When a debug exception is taken to an Exception level using AArch32, this field is set to indicate the event that caused the exception:

| MOE | Meaning |
|--------|--|
| 0b0001 | Breakpoint |
| 0b0011 | Software breakpoint (BKPT) instruction |
| 0b0101 | Vector catch |
| 0b1010 | Watchpoint |

Read-only view of the equivalent bit in the [DBGDSCRext](#).

Bits [1:0]

Reserved, RES0.

Accessing DBGDSCRint

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0001 | 0b000 |

```

if Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    return DBGDSCRint;
elseif PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
    elseif ELUsingAArch32(EL1) && DBGDSCRext.UDCCdis == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> !=
'00') then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDA> != '00') then
        AArch32.TakeHypTrapException(0x05);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDSCRint;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then

```

```

        UNDEFINED;
    else
        AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGDSCRint;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGDSCRint;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return DBGDSCRint;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRRXext, Debug OS Lock Data Transfer Register, Receive, External View

The DBGDTRRXext characteristics are:

Purpose

Used for save/restore of [DBGDTRRXint](#). It is a component of the Debug Communications Channel.

Configuration

AArch32 System register DBGDTRRXext bits [31:0] are architecturally mapped to AArch64 System register [OSDTRRX_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGDTRRXext are UNDEFINED.

Attributes

DBGDTRRXext is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Update DTRRX without side-effect | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

- Update DTRRX without side-effect.
- Writes to this register update the value in DTRRX and do not change RXfull.
- Reads of this register return the last value written to DTRRX and do not change RXfull.
- For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.
- The reset behavior of this field is:
- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRRXext

Arm deprecates reads and writes of DBGDTRRXext through the System register interface when the OS Lock is unlocked, [DBGOSLSR.OSLK](#) == 0.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    return DBGDTRRXext;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDTRRXext;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDTRRXext;
elseif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else

```

```
return DBGDTRRXext;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    DBGDTRRXext = R[t];
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDTRRXext = R[t];
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDTRRXext = R[t];
elseif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else

```

```
DBGDTRRXext = R[t];
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRRXint, Debug Data Transfer Register, Receive

The DBGDTRRXint characteristics are:

Purpose

Transfers data from an external debugger to the PE. For example, it is used by a debugger transferring commands and data to a debug target. See [DBGDTR_EL0](#) for additional architectural mappings. It is a component of the Debug Communications Channel.

Configuration

AArch32 System register DBGDTRRXint bits [31:0] are architecturally mapped to AArch64 System register [DBGDTRRX_EL0\[31:0\]](#).

AArch32 System register DBGDTRRXint bits [31:0] are architecturally mapped to External register [DBGDTRRX_EL0\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGDTRRXint are UNDEFINED.

Attributes

DBGDTRRXint is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Update DTRRX | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Update DTRRX.

Reads of this register:

- If RXfull is set to 1, return the last value written to DTRRX.
- If RXfull is set to 0, return an UNKNOWN value.

After the read, RXfull is cleared to 0.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRRXint

Data can be stored to memory from this register using STC.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| | | | | |
|--------|------|-----|-----|------|
| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0101 | 0b000 |
|--------|-------|--------|--------|-------|

```

if Halted() then
    return DBGDTRRXint;
elsif PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
        elsif ELUsingAArch32(EL1) && DBGDSCRExt.UCCdis == '1' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
                AArch32.TakeHypTrapException(0x05);
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> !=
'00') then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDA> != '00') then
                AArch32.TakeHypTrapException(0x05);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
                AArch32.TakeMonitorTrapException();
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            else
                return DBGDTRRXint;
        elsif PSTATE.EL == EL1 then
            if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
                AArch32.TakeHypTrapException(0x05);
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
                AArch32.TakeHypTrapException(0x05);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
                AArch32.TakeMonitorTrapException();
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            else
                return DBGDTRRXint;
        elsif PSTATE.EL == EL2 then
            if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
                AArch32.TakeMonitorTrapException();
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            else
                return DBGDTRRXint;
        elsif PSTATE.EL == EL3 then
            if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
                AArch32.TakeMonitorTrapException();
            else
                return DBGDTRRXint;

```


DBGDTRTXext, Debug OS Lock Data Transfer Register, Transmit

The DBGDTRTXext characteristics are:

Purpose

Used for save/restore of [DBGDTRTXint](#). It is a component of the Debug Communication Channel.

Configuration

AArch32 System register DBGDTRTXext bits [31:0] are architecturally mapped to AArch64 System register [OSDTRTX_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGDTRTXext are UNDEFINED.

Attributes

DBGDTRTXext is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Return DTRTX without side-effect | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Return DTRTX without side-effect.

Reads of this register return the value in DTRTX and do not change TXfull.

Writes of this register update the value in DTRTX and do not change TXfull.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRTXext

Arm deprecates reads and writes of DBGDTRTXext through the System register interface when the OS Lock is unlocked, [DBGOSLSR](#).OSLK == 0.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    return DBGDTRTText;
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDTRTText;
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGDTRTText;
elseif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else

```

```
return DBGDTRTText;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif Halted() && ConstrainUnpredictableBool(Unpredictable_IGNORETRAPINDEBUG) then
    DBGDTRTText = R[t];
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
        AArch32.TakeHypTrapException(0x05);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDTRTText = R[t];
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        UNDEFINED;
    elseif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGDTRTText = R[t];
elseif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
        AArch32.TakeMonitorTrapException();
    else

```

```
DBGDTRTText = R[t];
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRTXint, Debug Data Transfer Register, Transmit

The DBGDTRTXint characteristics are:

Purpose

Transfers data from the PE to an external debugger. For example, it is used by a debug target to transfer data to the debugger. See [DBGDTR_EL0](#) for additional architectural mappings. It is a component of the Debug Communication Channel.

Configuration

AArch32 System register DBGDTRTXint bits [31:0] are architecturally mapped to AArch64 System register [DBGDTRTX_EL0\[31:0\]](#).

AArch32 System register DBGDTRTXint bits [31:0] are architecturally mapped to External register [DBGDTRTX_EL0\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGDTRTXint are UNDEFINED.

Attributes

DBGDTRTXint is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Return DTRTX | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Return DTRTX.

Writes to this register:

- If TXfull is set to 1, set DTRTX to UNKNOWN.
- If TXfull is set to 0, update the value in DTRTX.

After the write, TXfull is set to 1.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRTXint

Data can be loaded from memory into this register using 'LDC (immediate)' and 'LDC (literal)'.

Accesses to this register use the following encodings in the System register encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| | | | | |
|--------|------|-----|-----|------|
| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0101 | 0b000 |
|--------|-------|--------|--------|-------|

```

if Halted() then
    DBGDTRTXint = R[t];
elsif PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && MDSCR_EL1.TDCC == '1' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x05);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x05);
        elsif ELUsingAArch32(EL1) && DBGDSCRExt.UDCCdis == '1' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
                AArch32.TakeHypTrapException(0x05);
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && (HCR_EL2.TGE == '1' || MDCR_EL2.<TDE,TDA> !=
'00') then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && (HCR.TGE == '1' || HDCR.<TDE,TDA> != '00') then
                AArch32.TakeHypTrapException(0x05);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
                AArch32.TakeMonitorTrapException();
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            else
                DBGDTRTXint = R[t];
        elsif PSTATE.EL == EL1 then
            if EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TDCC == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TDCC == '1' then
                AArch32.TakeHypTrapException(0x05);
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x05);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
                AArch32.TakeHypTrapException(0x05);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
                AArch32.TakeMonitorTrapException();
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            else
                DBGDTRTXint = R[t];
        elsif PSTATE.EL == EL2 then
            if HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDCC == '1' then
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TDCC == '1' then
                AArch32.TakeMonitorTrapException();
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            else
                DBGDTRTXint = R[t];
        elsif PSTATE.EL == EL3 then
            if PSTATE.M != M32_Monitor && SDCR.TDCC == '1' then
                AArch32.TakeMonitorTrapException();
            else
                DBGDTRTXint = R[t];

```


DBGOSDLR, Debug OS Double Lock Register

The DBGOSDLR characteristics are:

Purpose

Locks out the external debug interface.

Configuration

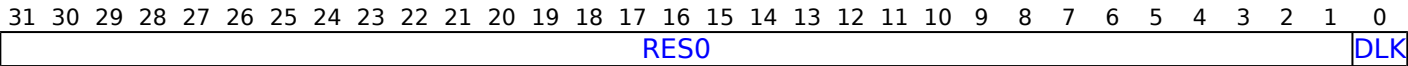
AArch32 System register DBGOSDLR bits [31:0] are architecturally mapped to AArch64 System register [OSDLR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGOSDLR are UNDEFINED.

Attributes

DBGOSDLR is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

DLK, bit [0]

When FEAT_DoubleLock is implemented:

OS Double Lock control bit.

| DLK | Meaning |
|-----|--|
| 0b0 | OS Double Lock unlocked. |
| 0b1 | OS Double Lock locked, if DBGPRCR .CORENPDRQ (Core no powerdown request) bit is set to 0 and the PE is in Non-debug state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RAZ/WI.

Accessing DBGOSDLR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| | | | | |
|--------|------|-----|-----|------|
| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0001 | 0b0011 | 0b100 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL2.TDOSA") then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDOSA> != '00' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by HDCR.TDOSA")
then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGOSDLR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGOSDLR;
elsif PSTATE.EL == EL3 then
    return DBGOSDLR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0001 | 0b0011 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL2.TDOSA") then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDOSA> != '00' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by HDCR.TDOSA")
then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            DBGOSDLR = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' &&
(IsFeatureImplemented(FEAT_DoubleLock) || boolean IMPLEMENTATION_DEFINED "Trapped by
MDCR_EL3.TDOSA") then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            DBGOSDLR = R[t];
elsif PSTATE.EL == EL3 then
    DBGOSDLR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGOSECCR, Debug OS Lock Exception Catch Control Register

The DBGOSECCR characteristics are:

Purpose

Provides a mechanism for an operating system to access the contents of [EDECCR](#) that are otherwise invisible to software, so it can save/restore the contents of [EDECCR](#) over powerdown on behalf of the external debugger.

Configuration

AArch32 System register DBGOSECCR bits [31:0] are architecturally mapped to AArch64 System register [OSECCR_EL1\[31:0\]](#).

AArch32 System register DBGOSECCR bits [31:0] are architecturally mapped to External register [EDECCR\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGOSECCR are UNDEFINED.

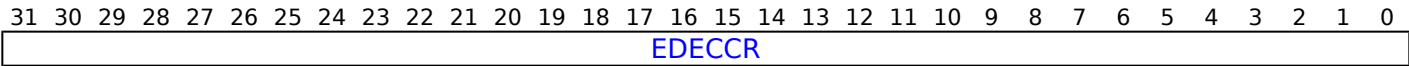
If [DBGOSLSR.OSLK](#) == 0 then DBGOSECCR returns an UNKNOWN value on reads and ignores writes.

Attributes

DBGOSECCR is a 32-bit register.

Field descriptions

When [DBGOSLSR.OSLK](#) == 1:



EDECCR, bits [31:0]

Used for save/restore to [EDECCR](#) over powerdown.

Reads or writes to this field are indirect accesses to [EDECCR](#).

Accessing DBGOSECCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' then
        return bits(32) UNKNOWN;
    else
        return DBGOSECCR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBGOSLSR.OSLK == '0' then
        return bits(32) UNKNOWN;
    else
        return DBGOSECCR;
elsif PSTATE.EL == EL3 then
    if DBGOSLSR.OSLK == '0' then
        return bits(32) UNKNOWN;
    else
        return DBGOSECCR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBG0SLSR.OSLK == '0' then
        //no operation
    else
        DBG0SECCR = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBG0SLSR.OSLK == '0' then
        //no operation
    else
        DBG0SECCR = R[t];
elsif PSTATE.EL == EL3 then
    if DBG0SLSR.OSLK == '0' then
        //no operation
    else
        DBG0SECCR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGOSLAR, Debug OS Lock Access Register

The DBGOSLAR characteristics are:

Purpose

Provides a lock for the debug registers. The OS Lock also disables some debug exceptions and debug events.

Configuration

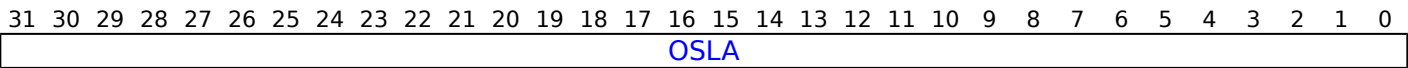
This register is present only when EL1 is capable of using AArch32. Otherwise, direct accesses to DBGOSLAR are UNDEFINED.

The OS Lock can also be locked or unlocked using the AArch64 System register [OSLAR_EL1](#) and External register [OSLAR_EL1](#).

Attributes

DBGOSLAR is a 32-bit register.

Field descriptions



OSLA, bits [31:0]

OS Lock Access. Writing the value 0xC5ACCE55 to the DBGOSLAR sets the OS Lock to 1. Writing any other value sets the OS Lock to 0.

Use [DBGOSLSR.OSLK](#) to check the current status of the lock.

Accessing DBGOSLAR

Accesses to this register use the following encodings in the System register encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0001 | 0b0000 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDOSA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            DBGOSLAR = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            DBGOSLAR = R[t];
    elsif PSTATE.EL == EL3 then
        DBGOSLAR = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGOSLSR, Debug OS Lock Status Register

The DBGOSLSR characteristics are:

Purpose

Provides status information for the OS Lock.

Configuration

AArch32 System register DBGOSLSR bits [31:0] are architecturally mapped to AArch64 System register [OSLSR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGOSLSR are UNDEFINED.

The OS Lock status is also visible in the external debug interface through EDPRSR.

Attributes

DBGOSLSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|-----|------|---------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | OSLM[1] | | nTT | OSLK | OSLM[0] | | | | | | | | | | | | | | |

Bits [31:4]

Reserved, RES0.

OSLM, bits [3, 0]

OS Lock model implemented. Identifies the form of OS save and restore mechanism implemented.

| OSLM | Meaning |
|------|--------------------------|
| 0b00 | OS Lock not implemented. |
| 0b10 | OS Lock implemented. |

All other values are reserved. In an Armv8 implementation the value 0b00 is not permitted.

The OSLM field is split as follows:

- OSLM[1] is DBGOSLSR[3].
- OSLM[0] is DBGOSLSR[0].

nTT, bit [2]

Not 32-bit access. This bit is always RAZ. It indicates that a 32-bit access is needed to write the key to the OS Lock Access Register.

OSLK, bit [1]

OS Lock Status.

| OSLK | Meaning |
|------|-------------------|
| 0b0 | OS Lock unlocked. |
| 0b1 | OS Lock locked. |

The OS Lock is locked and unlocked by writing to the OS Lock Access Register.

The reset behavior of this field is:

- On a Cold reset, this field resets to 1.

Accessing DBGOSLSR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0001 | 0b0001 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDOSA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        if Halted() && EDSR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGOSLSR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
            if Halted() && EDSR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            else
                return DBGOSLSR;
    elsif PSTATE.EL == EL3 then
        return DBGOSLSR;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGPRCR, Debug Power Control Register

The DBGPRCR characteristics are:

Purpose

Controls behavior of the PE on powerdown request.

Configuration

AArch32 System register DBGPRCR bits [31:0] are architecturally mapped to AArch64 System register [DBGPRCR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGPRCR are UNDEFINED.

Bit [0] of this register is mapped to [EDPRCR](#).CORENPDRQ, bit [0] of the external view of this register.

The other bits in these registers are not mapped to each other.

Attributes

DBGPRCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | CORENPDRQ | | | | | | | | | | | | | | | |

Bits [31:1]

Reserved, RES0.

CORENPDRQ, bit [0]

When FEAT_DoPD is implemented:

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

| CORENPDRQ | Meaning |
|-----------|--|
| 0b0 | If the system responds to a powerdown request, it powers down Core power domain. |
| 0b1 | If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain. |

In an implementation that includes the recommended external debug interface, this bit drives the DBGNOPWRDWN signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to the Cold reset value on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states see 'Core power domain power states'.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

On a Cold reset, if the powerup request is implemented and the powerup request has been asserted, this field is set to an IMPLEMENTATION DEFINED choice of 0 or 1. If the powerup request is not asserted, this field is set to 0.

Otherwise:

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

| CORENPDRQ | Meaning |
|-----------|--|
| 0b0 | If the system responds to a powerdown request, it powers down Core power domain. |
| 0b1 | If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain. |

In an implementation that includes the recommended external debug interface, this bit drives the DBGNOPWRDWN signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to the value of [EDPRCR.COREPURQ](#) on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states see 'Core power domain power states'.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The reset behavior of this field is:

- On a Cold reset, this field resets to the value in [EDPRCR.COREPURQ](#).

Accessing DBGPRCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0001 | 0b0100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDOSA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGPRCR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        return DBGPRCR;
elsif PSTATE.EL == EL3 then
    return DBGPRCR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0001 | 0b0100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDOSA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDOSA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGPRCR = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDOSA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGPRCR = R[t];
elsif PSTATE.EL == EL3 then
    DBGPRCR = R[t];

```


DBGVCR, Debug Vector Catch Register

The DBGVCR characteristics are:

Purpose

Controls Vector Catch debug events.

Configuration

AArch32 System register DBGVCR bits [31:0] are architecturally mapped to AArch64 System register [DBGVCR32_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGVCR are UNDEFINED.

This register is required in all implementations.

Attributes

DBGVCR is a 32-bit register.

Field descriptions

When EL3 is implemented and EL3 is using AArch32:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|-----|------|-----|-----|-----|-----|------|----|----|----|------|----|----|----|----|----|------|----|----|----|------|----|----|------|----|----|----|----|------|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NSF | NSI | RES0 | NSD | NSP | NSS | NSU | RES0 | | | | RES0 | | | | MF | MI | RES0 | MD | MP | MS | RES0 | SF | SI | RES0 | SD | SP | SS | SU | RES0 | | |

NSF, bit [31]

FIQ vector catch enable in Non-secure state.

The exception vector offset is 0x1C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSI, bit [30]

IRQ vector catch enable in Non-secure state.

The exception vector offset is 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [29]

Reserved, RES0.

NSD, bit [28]

Data Abort vector catch enable in Non-secure state.

The exception vector offset is 0x10.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSP, bit [27]

Prefetch Abort vector catch enable in Non-secure state.

The exception vector offset is 0x0C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSS, bit [26]

Supervisor Call (SVC) vector catch enable in Non-secure state.

The exception vector offset is 0x08.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSU, bit [25]

Undefined Instruction vector catch enable in Non-secure state.

The exception vector offset is 0x04.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [24:16]

Reserved, RES0.

MF, bit [15]

FIQ vector catch enable in Monitor mode.

The exception vector offset is 0x1C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MI, bit [14]

IRQ vector catch enable in Monitor mode.

The exception vector offset is 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

MD, bit [12]

Data Abort vector catch enable in Monitor mode.

The exception vector offset is 0x10.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MP, bit [11]

Prefetch Abort vector catch enable in Monitor mode.

The exception vector offset is 0x0C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

MS, bit [10]

Secure Monitor Call (SMC) vector catch enable in Monitor mode.

The exception vector offset is 0x08.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [9:8]

Reserved, RES0.

SF, bit [7]

FIQ vector catch enable in Secure state.

The exception vector offset is 0x1C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SI, bit [6]

IRQ vector catch enable in Secure state.

The exception vector offset is 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

SD, bit [4]

Data Abort vector catch enable in Secure state.

The exception vector offset is 0x10.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SP, bit [3]

Prefetch Abort vector catch enable in Secure state.

The exception vector offset is 0x0C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SS, bit [2]

Supervisor Call (SVC) vector catch enable in Secure state.

The exception vector offset is 0x08.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SU, bit [1]

Undefined Instruction vector catch enable in Secure state.

The exception vector offset is 0x04.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

When EL3 is implemented and EL3 is using AArch64:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|-----|------|-----|-----|-----|-----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|------|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NSF | NSI | RES0 | NSD | NSP | NSS | NSU | RES0 | | | | | | | | | | | | SF | SI | RES0 | SD | SP | SS | SU | RES0 | | | | | |

NSF, bit [31]

FIQ vector catch enable in Non-secure state.

The exception vector offset is 0x1C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSI, bit [30]

IRQ vector catch enable in Non-secure state.

The exception vector offset is 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [29]

Reserved, RES0.

NSD, bit [28]

Data Abort vector catch enable in Non-secure state.

The exception vector offset is 0x10.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSP, bit [27]

Prefetch Abort vector catch enable in Non-secure state.

The exception vector offset is 0x0C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSS, bit [26]

Supervisor Call (SVC) vector catch enable in Non-secure state.

The exception vector offset is 0x08.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSU, bit [25]

Undefined Instruction vector catch enable in Non-secure state.

The exception vector offset is 0x04.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [24:8]

Reserved, RES0.

SF, bit [7]

FIQ vector catch enable in Secure state.

The exception vector offset is 0x1C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SI, bit [6]

IRQ vector catch enable in Secure state.

The exception vector offset is 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

SD, bit [4]

Data Abort vector catch enable in Secure state.

The exception vector offset is 0x10.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SP, bit [3]

Prefetch Abort vector catch enable in Secure state.

The exception vector offset is 0x0C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SS, bit [2]

Supervisor Call (SVC) vector catch enable in Secure state.

The exception vector offset is 0x08.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SU, bit [1]

Undefined Instruction vector catch enable in Secure state.

The exception vector offset is 0x04.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

When EL3 is not implemented:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|------|---|---|---|---|------|
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | F | I | RES0 | D | P | S | U | RES0 |

Bits [31:8]

Reserved, RES0.

F, bit [7]

FIQ vector catch enable.

The exception vector offset is 0x1C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [6]

IRQ vector catch enable.

The exception vector offset is 0x18.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

D, bit [4]

Data Abort vector catch enable.

The exception vector offset is 0x10.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P, bit [3]

Prefetch Abort vector catch enable.

The exception vector offset 0x0C.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [2]

Supervisor Call (SVC) vector catch enable.

The exception vector offset is 0x08.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

U, bit [1]

Undefined Instruction vector catch enable.

The exception vector offset is 0x04.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

Accessing DBGVCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0111 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGVCR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGVCR;
    elsif PSTATE.EL == EL3 then
        return DBGVCR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0111 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            DBGVCR = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            DBGVCR = R[t];
    elsif PSTATE.EL == EL3 then
        DBGVCR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGWCR<n>, Debug Watchpoint Control Registers, n = 0 - 15

The DBGWCR<n> characteristics are:

Purpose

Holds control information for a watchpoint. Forms watchpoint n together with value register [DBGWVR<n>](#).

Configuration

AArch32 System register DBGWCR<n> bits [31:0] are architecturally mapped to AArch64 System register [DBGWCR<n>_EL1\[31:0\]](#).

AArch32 System register DBGWCR<n> bits [31:0] are architecturally mapped to External register [DBGWCR<n>_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGWCR<n> are UNDEFINED.

If watchpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGWCR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

When the E field is zero, all the other fields in the register are ignored.

Bits [31:29]

Reserved, RES0.

MASK, bits [28:24]

Address mask. Only objects up to 2GB can be watched using a single mask.

| MASK | Meaning |
|---------|-----------|
| 0b00000 | No mask. |
| 0b00001 | Reserved. |
| 0b00010 | Reserved. |

If programmed with a reserved value, a watchpoint must behave as if either:

- MASK has been programmed with a defined value, which might be 0 (no mask), other than for a direct read of DBGWCRn_EL1.
- The watchpoint is disabled.

Software must not rely on this property because the behavior of reserved values might change in a future revision of the architecture.

Other values mask the corresponding number of address bits, from 0b00011 masking 3 address bits (0x00000007 mask for address) to 0b11111 masking 31 address bits (0x7FFFFFFF mask for address).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [23:21]

Reserved, RES0.

WT, bit [20]

Watchpoint type. Possible values are:

| WT | Meaning |
|-----|------------------------------|
| 0b0 | Unlinked data address match. |
| 0b1 | Linked data address match. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked breakpoint number. For Linked data address watchpoints, this specifies the index of the Context-matching breakpoint linked to.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the HMC and PAC fields.

For more information, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions', and 'Reserved DBGWCR<n>.{SSC, HMC, PAC} values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and PAC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

BAS, bits [12:5]

Byte address select. Each bit of this field selects whether a byte from within the word or double-word addressed by [DBGWVR<n>](#) is being watched.

| BAS | Description |
|------------|--|
| 0bxxxxxxx1 | Match byte at DBGWVR<n> |
| 0bxxxxxx1x | Match byte at DBGWVR<n> +1 |
| 0bxxxxx1xx | Match byte at DBGWVR<n> +2 |
| 0bxxxx1xxx | Match byte at DBGWVR<n> +3 |

In cases where [DBGWVR<n>](#) addresses a double-word:

| BAS | Description, if DBGWVR<n>[2] == 0 |
|------------|---|
| 0bxxx1xxxx | Match byte at DBGWVR<n>+4 |
| 0bxx1xxxxx | Match byte at DBGWVR<n>+5 |
| 0bx1xxxxxx | Match byte at DBGWVR<n>+6 |
| 0b1xxxxxxx | Match byte at DBGWVR<n>+7 |

If [DBGWVR<n>\[2\] == 1](#), only BAS[3:0] are used and BAS[7:4] are ignored. Arm deprecates setting [DBGWVR<n>\[2\] == 1](#).

The valid values for BAS are non-zero binary numbers all of whose set bits are contiguous. All other values are reserved and must not be used by software. See 'Reserved DBGWCR<n>.BAS values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LSC, bits [4:3]

Load/store control. This field enables watchpoint matching on the type of access being made. Possible values of this field are:

| LSC | Meaning |
|------|---|
| 0b01 | Match instructions that load from a watchpointed address. |
| 0b10 | Match instructions that store to a watchpointed address. |
| 0b11 | Match instructions that load from or store to a watchpointed address. |

All other values are reserved, but must behave as if the watchpoint is disabled. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PAC, bits [2:1]

Privilege of access control. Determines the Exception level or levels at which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and HMC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable watchpoint n. Possible values are:

| E | Meaning |
|-----|----------------------|
| 0b0 | Watchpoint disabled. |
| 0b1 | Watchpoint enabled. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGWCR<n>

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | n[3:0] | 0b111 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            return DBGWCR[UInt(CRm<3:0>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        elsif DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            return DBGWCR[UInt(CRm<3:0>)];
    elsif PSTATE.EL == EL3 then
        if DBGOSLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            return DBGWCR[UInt(CRm<3:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | n[3:0] | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWCR[UInt(CRm<3:0>)] = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    elsif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWCR[UInt(CRm<3:0>)] = R[t];
elsif PSTATE.EL == EL3 then
    if DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
        Halt(DebugHalt_SoftwareAccess);
    else
        DBGWCR[UInt(CRm<3:0>)] = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGWFAR, Debug Watchpoint Fault Address Register

The DBGWFAR characteristics are:

Purpose

Previously returned information about the address of the instruction that accessed a watchpointed address. Is now deprecated and RES0.

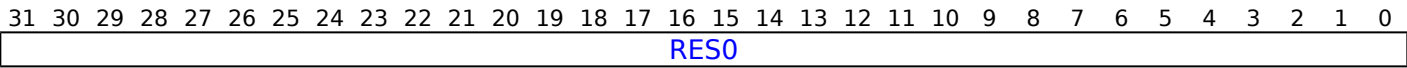
Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGWFAR are UNDEFINED.

Attributes

DBGWFAR is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing DBGWFAR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGWFAR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        else
            return DBGWFAR;
    elsif PSTATE.EL == EL3 then
        return DBGWFAR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGWFAR = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
    else
        DBGWFAR = R[t];
    elsif PSTATE.EL == EL3 then
        DBGWFAR = R[t];

```


DBGWVR<n>, Debug Watchpoint Value Registers, n = 0 - 15

The DBGWVR<n> characteristics are:

Purpose

Holds a data address value for use in watchpoint matching. Forms watchpoint n together with control register [DBGWCR<n>](#).

Configuration

AArch32 System register DBGWVR<n> bits [31:0] are architecturally mapped to AArch64 System register [DBGWVR<n>_EL1\[31:0\]](#).

AArch32 System register DBGWVR<n> bits [31:0] are architecturally mapped to External register [DBGWVR<n>_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DBGWVR<n> are UNDEFINED.

If watchpoint n is not implemented then accesses to this register are UNDEFINED.

Attributes

DBGWVR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |

VA, bits [31:2]

Bits[31:2] of the address value for comparison.

Arm deprecates setting [DBGWVR<n>](#)[2] == 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

Accessing DBGWVR<n>

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | n[3:0] | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        elsif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            return DBGWVR[UInt(CRm<3:0>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            elsif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
                Halt(DebugHalt_SoftwareAccess);
            else
                return DBGWVR[UInt(CRm<3:0>)];
    elsif PSTATE.EL == EL3 then
        if DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            return DBGWVR[UInt(CRm<3:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b000 | 0b0000 | n[3:0] | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.<TDE,TDA> != '00' then
        AArch32.TakeHypTrapException(0x05);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x05);
        elsif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            DBGWVR[UInt(CRm<3:0>)] = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x05);
            elsif DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
                Halt(DebugHalt_SoftwareAccess);
            else
                DBGWVR[UInt(CRm<3:0>)] = R[t];
    elsif PSTATE.EL == EL3 then
        if DBG0SLSR.OSLK == '0' && HaltingAllowed() && EDSCR.TDA == '1' then
            Halt(DebugHalt_SoftwareAccess);
        else
            DBGWVR[UInt(CRm<3:0>)] = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCCIMVAC, Data Cache line Clean and Invalidate by VA to PoC

The DCCIMVAC characteristics are:

Purpose

Clean and Invalidate data or unified cache line by virtual address to PoC.

Configuration

AArch32 System instruction DCCIMVAC performs the same function as AArch64 System instruction [DC CIVAC](#).

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to DCCIMVAC are UNDEFINED.

Attributes

DCCIMVAC is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DCCIMVAC instruction

Execution of this instruction might require an address translation from VA to PA, and that translation might fault.

If FEAT_CMOW is implemented, [HCRX_EL2](#).CMOW is 1, and EL1 or EL0 access is enabled, when executed at EL1 or EL0, the instruction has stage 2 read permission to the VA but does not have stage 2 write permission to the VA, the instruction generates a stage 2 Permission fault.

For more information, see 'AArch32 data cache maintenance instructions (DC*)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TPC == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_CleanInvalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_CleanInvalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_CleanInvalidate, CacheOpScope_PoC);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCCISW, Data Cache line Clean and Invalidate by Set/Way

The DCCISW characteristics are:

Purpose

Clean and Invalidate data or unified cache line by set/way.

Configuration

AArch32 System instruction DCCISW performs the same function as AArch64 System instruction [DC C1SW](#).

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to DCCISW are UNDEFINED.

Attributes

DCCISW is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|------|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SetWay | | | | | | | | | | | | | | | | Level | | | RES0 | | | | | | | | | | | | |

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

$A = \text{Log}_2(\text{ASSOCIATIVITY})$, $L = \text{Log}_2(\text{LINELEN})$, $B = (L + S)$, $S = \text{Log}_2(\text{NSETS})$.

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing the DCCISW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED
- The instruction performs cache maintenance on one of:
 - No cache lines.
 - A single arbitrary cache line.

- Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TSW == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_CleanInvalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_CleanInvalidate, CacheOpScope_SetWay);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCCMVAC, Data Cache line Clean by VA to PoC

The DCCMVAC characteristics are:

Purpose

Clean data or unified cache line by virtual address to PoC.

Configuration

AArch32 System instruction DCCMVAC performs the same function as AArch64 System instruction [DC CVAC](#).

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to DCCMVAC are UNDEFINED.

Attributes

DCCMVAC is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DCCMVAC instruction

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'AArch32 data cache maintenance instructions (DC*)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TPC == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_PoC);
elseif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_PoC);
elseif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_PoC);

```


DCCMVAU, Data Cache line Clean by VA to PoU

The DCCMVAU characteristics are:

Purpose

Clean data or unified cache line by virtual address to PoU.

Configuration

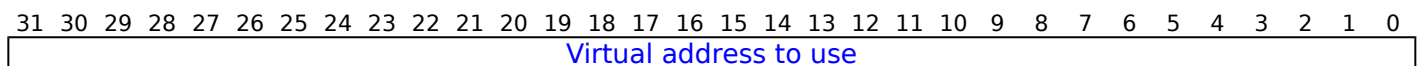
AArch32 System instruction DCCMVAU performs the same function as AArch64 System instruction [DC CVAU](#).

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to DCCMVAU are UNDEFINED.

Attributes

DCCMVAU is a 32-bit System instruction.

Field descriptions



Bits [31:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DCCMVAU instruction

Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'AArch32 data cache maintenance instructions (DC*)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TOCU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TPU == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TOCU == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_PoU);
elsif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_PoU);
elsif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_PoU);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCCSW, Data Cache line Clean by Set/Way

The DCCSW characteristics are:

Purpose

Clean data or unified cache line by set/way.

Configuration

AArch32 System instruction DCCSW performs the same function as AArch64 System instruction [DC CSW](#).

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to DCCSW are UNDEFINED.

Attributes

DCCSW is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|-------|------|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SetWay | | | | | | | | | | | | | | | | | | | | | | | | | | | | Level | RES0 | | |

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

$A = \text{Log}_2(\text{ASSOCIATIVITY})$, $L = \text{Log}_2(\text{LINELEN})$, $B = (L + S)$, $S = \text{Log}_2(\text{NSETS})$.

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing the DCCSW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED
- The instruction performs cache maintenance on one of:
 - No cache lines.
 - A single arbitrary cache line.
 - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b1010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TSW == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_Clean, CacheOpScope_SetWay);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCIMVAC, Data Cache line Invalidate by VA to PoC

The DCIMVAC characteristics are:

Purpose

Invalidate data or unified cache line by virtual address to PoC.

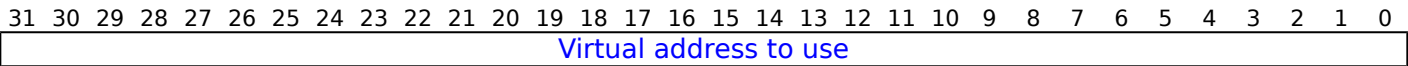
Configuration

AArch32 System instruction DCIMVAC performs the same function as AArch64 System instruction [DC IVAC](#).
This instruction is present only when AArch32 is supported. Otherwise, direct accesses to DCIMVAC are UNDEFINED.

Attributes

DCIMVAC is a 32-bit System instruction.

Field descriptions



Bits [31:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the DCIMVAC instruction

It is IMPLEMENTATION DEFINED whether, when this instruction is executed, it can generate a watchpoint. If this instruction can generate a watchpoint this is prioritized in the same way as other watchpoints.
Execution of this instruction might require an address translation from VA to PA, and that translation might fault. For more information, see 'AArch32 data cache maintenance instructions (DC*)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b0110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPCP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TPC == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_Invalidate, CacheOpScope_PoC);
elsif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_Invalidate, CacheOpScope_PoC);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DCISW, Data Cache line Invalidate by Set/Way

The DCISW characteristics are:

Purpose

Invalidate data or unified cache line by set/way.

Configuration

AArch32 System instruction DCISW performs the same function as AArch64 System instruction [DC ISW](#).

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to DCISW are UNDEFINED.

Attributes

DCISW is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|------|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SetWay | | | | | | | | | | | | | | | | Level | | | RES0 | | | | | | | | | | | | |

SetWay, bits [31:4]

Contains two fields:

- Way, bits[31:32-A], the number of the way to operate on.
- Set, bits[B-1:L], the number of the set to operate on.

Bits[L-1:4] are RES0.

$A = \text{Log}_2(\text{ASSOCIATIVITY})$, $L = \text{Log}_2(\text{LINELEN})$, $B = (L + S)$, $S = \text{Log}_2(\text{NSETS})$.

ASSOCIATIVITY, LINELEN (line length, in bytes), and NSETS (number of sets) have their usual meanings and are the values for the cache level being operated on. The values of A and S are rounded up to the next integer.

Level, bits [3:1]

Cache level to operate on, minus 1. For example, this field is 0 for operations on L1 cache, or 1 for operations on L2 cache.

Bit [0]

Reserved, RES0.

Executing the DCISW instruction

If this instruction is executed with a set, way or level argument that is larger than the value supported by the implementation then the behavior is CONSTRAINED UNPREDICTABLE and one of the following occurs:

- The instruction is UNDEFINED
- The instruction performs cache maintenance on one of:
 - No cache lines.
 - A single arbitrary cache line.
 - Multiple arbitrary cache lines.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b0110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TSW == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TSW == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.DC(R[t], CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL2 then
    AArch32.DC(R[t], CacheOp_Invalidate, CacheOpScope_SetWay);
elsif PSTATE.EL == EL3 then
    AArch32.DC(R[t], CacheOp_Invalidate, CacheOpScope_SetWay);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DFAR, Data Fault Address Register

The DFAR characteristics are:

Purpose

Holds the virtual address of the faulting address that caused a synchronous Data Abort exception.

Configuration

AArch32 System register DFAR bits [31:0] are architecturally mapped to AArch64 System register [FAR_EL1\[31:0\]](#).

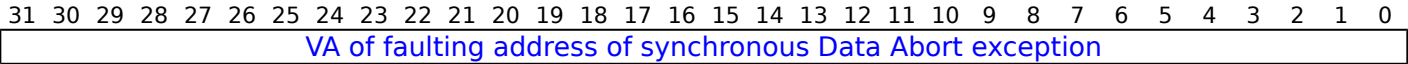
AArch32 System register DFAR bits [31:0] (S) are architecturally mapped to AArch32 System register [HDFAR\[31:0\]](#) when EL2 is implemented, EL3 is implemented and the implementation only supports execution in AArch32 state.

This register is present only when AArch32 is supported. Otherwise, direct accesses to DFAR are UNDEFINED.

Attributes

DFAR is a 32-bit register.

Field descriptions



Bits [31:0]

VA of faulting address of synchronous Data Abort exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DFAR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return DFAR_NS;
    else
        return DFAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return DFAR_NS;
    else
        return DFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return DFAR_S;
    else
        return DFAR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        DFAR_NS = R[t];
    else
        DFAR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        DFAR_NS = R[t];
    else
        DFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        DFAR_S = R[t];
    else
        DFAR_NS = R[t];

```

DFSR, Data Fault Status Register

The DFSR characteristics are:

Purpose

Holds status information about the last data fault.

Configuration

AArch32 System register DFSR bits [31:0] are architecturally mapped to AArch64 System register [ESR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DFSR are UNDEFINED.

The current translation table format determines which format of the register is used.

Attributes

DFSR is a 32-bit register.

Field descriptions

When TTBCR.EAE == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-------|-----|-------|------|------|--------|---|---|---------|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | FnV | AET | CMExt | WnR | FS[4] | LPAE | RES0 | Domain | | | FS[3:0] | | | | | | |

Bits [31:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|--|
| 0b0 | DFAR is valid. |
| 0b1 | DFAR is not valid, and holds an UNKNOWN value. |

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Data Abort exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AET, bits [15:14]

When FEAT_RAS is implemented:

Asynchronous Error Type. When DFSC is 0b010001, describes the PE error state after taking the SError interrupt exception. Possible values are:

| AET | Meaning |
|------|----------------------------|
| 0b00 | Uncontainable (UC). |
| 0b01 | Unrecoverable state (UEU). |
| 0b10 | Restartable state (UEO). |
| 0b11 | Recoverable state (UER). |

This field is valid only if the DFSC code is 0b010001. It is RES0 for all other aborts.

In the event of multiple errors taken as a single SError interrupt exception, the overall PE error state is reported.

Note

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CM, bit [13]

Cache maintenance fault. For synchronous faults, this bit indicates whether a cache maintenance instruction generated the fault.

| CM | Meaning |
|-----|---|
| 0b0 | Abort not caused by execution of a cache maintenance instruction. |
| 0b1 | Abort caused by execution of a cache maintenance instruction, or on an address translation. |

On a synchronous Data Abort on a translation table walk, this bit is UNKNOWN.

On an asynchronous fault, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [11]

Write not Read bit. Indicates whether the abort was caused by a write or a read instruction.

| WnR | Meaning |
|-----|--------------------------------------|
| 0b0 | Abort caused by a read instruction. |
| 0b1 | Abort caused by a write instruction. |

For faults on the cache maintenance and address translation System instructions in the (coproc==0b1111) encoding space this bit always returns a value of 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FS, bits [10, 3:0]

Fault status bits. Possible values of FS[4:0] are:

| FS | Meaning | Applies when |
|-----------|--|----------------------------------|
| 0b000001 | Alignment fault. | |
| 0b000010 | Debug exception. | |
| 0b000011 | Access flag fault, level 1. | |
| 0b000100 | Fault on instruction cache maintenance. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Access flag fault, level 2. | |
| 0b000111 | Translation fault, level 2. | |
| 0b01000 | Synchronous External abort, not on translation table walk. | |
| 0b01001 | Domain fault, level 1. | |
| 0b01011 | Domain fault, level 2. | |
| 0b01100 | Synchronous External abort, on translation table walk, level 1. | |
| 0b01101 | Permission fault, level 1. | |
| 0b01110 | Synchronous External abort, on translation table walk, level 2. | |
| 0b01111 | Permission fault, level 2. | |
| 0b10000 | TLB conflict abort. | |
| 0b10100 | IMPLEMENTATION DEFINED fault (Lockdown fault). | |
| 0b10101 | IMPLEMENTATION DEFINED fault (Unsupported Exclusive access fault). | |
| 0b10110 | SError interrupt. | |
| 0b11000 | SError interrupt, from a parity or ECC error on memory access. | When FEAT_RAS is not implemented |
| 0b11001 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b11100 | Synchronous parity or ECC error on translation table walk, level 1. | When FEAT_RAS is not implemented |
| 0b11110 | Synchronous parity or ECC error on translation table walk, level 2. | When FEAT_RAS is not implemented |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Short-descriptor translation table lookup'.

The FS field is split as follows:

- FS[4] is DFSR[10].
- FS[3:0] is DFSR[3:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

| LPAE | Meaning |
|-------------|---|
| 0b0 | Using the Short-descriptor translation table formats. |
| 0b1 | Using the Long-descriptor translation table formats. |

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

Domain, bits [7:4]

The domain of the fault address.

Arm deprecates any use of this field, see 'The Domain field in the DFSR'.

This field is UNKNOWN for certain faults where the DFSR is updated and reported using the Short-descriptor FSR encodings, see 'Validity of Domain field on faults that update the DFSR when using the Short-descriptor encodings'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When TTBCR.EAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|----|--------|------|------|------|--------|---|---|---|---|---|---|---|---|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| RES0 | | | | | | | | | | | | | | | | FnV | AET | CM | ExtWnR | RES0 | LPAE | RES0 | STATUS | | | | | | | | | |

Bits [31:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|--|
| 0b0 | DFAR is valid. |
| 0b1 | DFAR is not valid, and holds an UNKNOWN value. |

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Data Abort exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AET, bits [15:14]

When FEAT_RAS is implemented:

Asynchronous Error Type. When DFSC is 0b010001, describes the PE error state after taking the SError interrupt exception. Possible values are:

| AET | Meaning |
|------|----------------------------|
| 0b00 | Uncontainable (UC). |
| 0b01 | Unrecoverable state (UEU). |
| 0b10 | Restartable state (UEO). |
| 0b11 | Recoverable state (UER). |

This field is valid only if the DFSC code is 0b010001. It is RES0 for all other aborts.

In the event of multiple errors taken as a single SError interrupt exception, the overall PE error state is reported.

Note

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CM, bit [13]

Cache maintenance fault. For synchronous faults, this bit indicates whether a cache maintenance instruction generated the fault.

| CM | Meaning |
|-----|---|
| 0b0 | Abort not caused by execution of a cache maintenance instruction. |
| 0b1 | Abort caused by execution of a cache maintenance instruction. |

On a synchronous Data Abort on a translation table walk, this bit is UNKNOWN.

On an asynchronous fault, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [11]

Write not Read bit. Indicates whether the abort was caused by a write or a read instruction.

| WnR | Meaning |
|-----|--------------------------------------|
| 0b0 | Abort caused by a read instruction. |
| 0b1 | Abort caused by a write instruction. |

For faults on the cache maintenance and address translation System instructions in the (coproc==0b1111) encoding space this bit always returns a value of 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [10]

Reserved, RES0.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

| LPAE | Meaning |
|-------------|---|
| 0b0 | Using the Short-descriptor translation table formats. |
| 0b1 | Using the Long-descriptor translation table formats. |

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status bits. Possible values of this field are:

| STATUS | Meaning | Applies when |
|----------|--|---|
| 0b000000 | Address size fault in translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010000 | Synchronous External abort, not on translation table walk. | |
| 0b010001 | Asynchronous SError interrupt. | |
| 0b010101 | Synchronous External abort on translation table walk, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk, level 3. | |
| 0b011000 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b011001 | Asynchronous SError interrupt, from a parity or ECC error on memory access. | When FEAT_RAS is not implemented |
| 0b011101 | Synchronous parity or ECC error on memory access on translation table walk, level 1. | When FEAT_RAS is not implemented |
| 0b011110 | Synchronous parity or ECC error on memory access on translation table walk, level 2. | When FEAT_RAS is not implemented |
| 0b011111 | Synchronous parity or ECC error on memory access on translation table walk, level 3. | When FEAT_RAS is not implemented |
| 0b100001 | Alignment fault. | |
| 0b100010 | Debug exception. | |
| 0b110000 | TLB conflict abort. | |
| 0b110100 | IMPLEMENTATION DEFINED fault (Lockdown). | |
| 0b110101 | IMPLEMENTATION DEFINED fault (Unsupported Exclusive access). | |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Long-descriptor translation table lookup'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DFSR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return DFSR_NS;
    else
        return DFSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return DFSR_NS;
    else
        return DFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return DFSR_S;
    else
        return DFSR_NS;

```

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        DFSR_NS = R[t];
    else
        DFSR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        DFSR_NS = R[t];
    else
        DFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        DFSR_S = R[t];
    else
        DFSR_NS = R[t];

```


DISR, Deferred Interrupt Status Register

The DISR characteristics are:

Purpose

Records that an SError interrupt has been consumed by an ESB instruction.

Configuration

AArch32 System register DISR bits [31:0] are architecturally mapped to AArch64 System register [DISR_EL1\[31:0\]](#) when the highest implemented Exception level is using AArch64.

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to DISR are UNDEFINED.

Attributes

DISR is a 32-bit register.

Field descriptions

When the ESB instruction is executed at EL2:

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|------|---|---|------|---|---|---|---|---|---|--|
| A | RES0 | | | | | | | | | | | | | | | | | | | AET | EA | RES0 | | | DFSC | | | | | | | |

A, bit [31]

Set to 1 when an ESB instruction defers an asynchronous SError interrupt. If the implementation does not include any sources of SError interrupt that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:12]

Reserved, RES0.

AET, bits [11:10]

Asynchronous Error Type. See the description of [HSR.AET](#) for an SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort Type. See the description of [HSR.EA](#) for an SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

DFSC, bits [5:0]

Fault Status Code. See the description of [HSR.DFSC](#) for an SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When the ESB instruction is executed at EL0 or EL1 and where TTBCR.EAE == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|----|----|----|----|----|----|-----|------|-----|------|-------|------|------|----|----|----|----|---------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| A | RES0 | | | | | | | | | | AET | RES0 | EXT | RES0 | FS[4] | LPAE | RES0 | | | | | FS[3:0] | | | | | | | | | |

A, bit [31]

Set to 1 when an ESB instruction defers an asynchronous SError interrupt. If the implementation does not include any sources of SError interrupt that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

Asynchronous Error Type. See the description of [DFSR.AET](#) for an SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

External abort Type. See the description of [DFSR.ExT](#) for an SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

FS, bits [10, 3:0]

Fault Status Code. See the description of [DFSR.FS](#) for an SError interrupt.

The FS field is split as follows:

- FS[4] is DISR[10].

- FS[3:0] is DISR[3:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

Format.

| LPAE | Meaning |
|------|--|
| 0b0 | Using the Short-descriptor translation table format. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:4]

Reserved, RES0.

When the ESB instruction is executed at EL0 or EL1 and where TTBCR.EAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|----|----|----|----|----|----|-----|------|-----|------|------|------|--------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| A | RES0 | | | | | | | | | | AET | RES0 | ExT | RES0 | LPAE | RES0 | STATUS | | | | | | | | | | | | | | |

A, bit [31]

Set to 1 when an ESB instruction defers an asynchronous SError interrupt. If the implementation does not include any sources of SError interrupt that can be synchronized by an Error Synchronization Barrier, then this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

Asynchronous Error Type. See the description of [DFSR.AET](#) for an SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

External abort Type. See the description of [DFSR.ExT](#) for an SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

Format.

| LPAE | Meaning |
|------|---|
| 0b1 | Using the Long-descriptor translation table format. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault Status Code. See the description of [DFSR.FS](#) for an SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DISR

An indirect write to DISR made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of DISR occurring in program order after the ESB instruction.

DISR is RAZ/WI if EL3 is implemented, the PE is in Non-debug state, and any of the following apply:

- EL3 is using AArch64, [SCR_EL3.EA](#) == 1, and any of the following apply:
 - The PE is executing at EL2.
 - The PE is executing at EL1 and (([SCR_EL3.NS](#) == 0 && [SCR_EL3.EEL2](#) == 0) || [HCR_EL2.AMO](#) == 0).
- EL3 is using AArch32, [SCR.EA](#) == 1, and any of the following apply:
 - The PE is executing at EL2.
 - The PE is executing at EL1 and ([SCR.NS](#) == 0 || [HCR.AMO](#) == 0).

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
        return VDISR_EL2;
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.AMO == '1' then
        return VDISR;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return Zeros();
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && !Halted() && SCR.EA == '1' then
        return Zeros();
    else
        return DISR;
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return Zeros();
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && !Halted() && SCR.EA == '1' then
        return Zeros();
    else
        return DISR;
elseif PSTATE.EL == EL3 then
    return DISR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
        VDISR_EL2 = R[t];
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.AMO == '1' then
        VDISR = R[t];
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && !Halted() && SCR_EL3.EA == '1' then
        //no operation
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && !Halted() && SCR.EA == '1' then
        //no operation
    else
        DISR = R[t];
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && !Halted() && SCR_EL3.EA == '1' then
        //no operation
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && !Halted() && SCR.EA == '1' then
        //no operation
    else
        DISR = R[t];
elseif PSTATE.EL == EL3 then
    DISR = R[t];

```

DLR, Debug Link Register

The DLR characteristics are:

Purpose

In Debug state, holds the address to restart from.

Configuration

AArch32 System register DLR bits [31:0] are architecturally mapped to AArch64 System register [DLR_EL0\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DLR are UNDEFINED.

Attributes

DLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Restart address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Restart address.

Accessing DLR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b011 | 0b0100 | 0b0101 | 0b001 |

```
if !Halted() then
    UNDEFINED;
else
    return DLR;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b011 | 0b0100 | 0b0101 | 0b001 |

```
if !Halted() then
    UNDEFINED;
else
    DLR = R[t];
```


DSPSR, Debug Saved Program Status Register

The DSPSR characteristics are:

Purpose

Holds the saved process state for Debug state. On entering Debug state, PSTATE information is written to this register. On exiting Debug state, values are copied from this register to PSTATE.

Configuration

AArch32 System register DSPSR bits [31:0] are architecturally mapped to AArch64 System register [DSPSR_EL0\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to DSPSR are UNDEFINED.

Attributes

DSPSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|---------|-----|----|------|----|----|----|----|----|---------|----|----|----|----|----|----|----|--------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| N | Z | C | V | Q | IT[1:0] | DIT | SS | SPAN | SS | IL | | GE | | IT[7:2] | | E | A | I | F | T | | M[4:0] | | | | | | | | | |

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on entering Debug state, and copied to PSTATE.N on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on entering Debug state, and copied to PSTATE.Z on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on entering Debug state, and copied to PSTATE.C on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on entering Debug state, and copied to PSTATE.V on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on entering Debug state, and copied to PSTATE.Q on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on entering Debug state, and copied to PSTATE.IT on exiting Debug state.

DSPSR.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is DSPSR[26:25].
- IT[7:2] is DSPSR[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIT, bit [24]

When FEAT_DIT is implemented:

Data Independent Timing. Set to the value of PSTATE.DIT on entering Debug state, and copied to PSTATE.DIT on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on entering Debug state, and copied to PSTATE.SSBS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on entering Debug state, and copied to PSTATE.PAN on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SS, bit [21]

Software Step. Set to the value of PSTATE.SS on entering Debug state, and conditionally copied to PSTATE.SS on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on entering Debug state, and copied to PSTATE.IL on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on entering Debug state, and copied to PSTATE.GE on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on entering Debug state, and copied to PSTATE.E on exiting Debug state.

If the implementation does not support big-endian operation, DSPSR.E is RES0. If the implementation does not support little-endian operation, DSPSR.E is RES1. On exiting Debug state, if the implementation does not support big-endian operation at the Exception level being returned to, DSPSR.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, DSPSR.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

Error interrupt mask. Set to the value of PSTATE.A on entering Debug state, and copied to PSTATE.A on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on entering Debug state, and copied to PSTATE.I on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on entering Debug state, and copied to PSTATE.F on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on entering Debug state, and copied to PSTATE.T on exiting Debug state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on entering Debug state, and copied to PSTATE.M[4:0] on exiting Debug state.

| M[4:0] | Meaning |
|---------------|----------------|
| 0b10000 | User. |
| 0b10001 | FIQ. |
| 0b10010 | IRQ. |
| 0b10011 | Supervisor. |
| 0b10110 | Monitor. |
| 0b10111 | Abort. |
| 0b11010 | Hyp. |
| 0b11011 | Undefined. |
| 0b11111 | System. |

Other values are reserved. If DPSR.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, exiting Debug state is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing DPSR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b011 | 0b0100 | 0b0101 | 0b000 |

```
if !Halted() then
    UNDEFINED;
else
    return DPSR;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b011 | 0b0100 | 0b0101 | 0b000 |

```
if !Halted() then
    UNDEFINED;
else
    DSPSR = R[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DTLBIALL, Data TLB Invalidate All

The DTLBIALL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from data TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at EL1, all entries that:
 - Would be required for the EL1&0 translation regime.
 - Match the current VMID, if EL2 is implemented and enabled in the current Security state.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at EL2, and if EL2 is enabled in the current Security state, the stage 1 or stage 2 translation table entries that would be required for the Non-secure PL1&0 translation regime and matches the current VMID.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to DTLBIALL are UNDEFINED.

Attributes

DTLBIALL is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the DTLBIALL instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
        && IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
            AArch32.DTLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_NSH,
            TLBI_ExcludeXS);
        else
            AArch32.DTLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_NSH, TLBI_AllAttr);
    endif PSTATE.EL == EL2 then
        AArch32.DTLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_NSH, TLBI_AllAttr);
    elsif PSTATE.EL == EL3 then
        AArch32.DTLBI_ALL(SecurityStateAtEL(EL3), Regime_EL30, Shareability_NSH, TLBI_AllAttr);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DTLBIASID, Data TLB Invalidate by ASID match

The DTLBIASID characteristics are:

Purpose

Invalidate all cached copies of translation table entries from data TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
 - Is from a level of lookup above the final level.
 - Is a non-global entry from the final level of lookup.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to DTLBIASID are UNDEFINED.

Attributes

DTLBIASID is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | ASID | | | | | | | |

Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.

Executing the DTLBIASID instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| | | | | |
|--------|------|-----|-----|------|
| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0110 | 0b010 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
        && IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
            AArch32.DTLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBI_ExcludeXS, R[t]);
        else
            AArch32.DTLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBI_AllAttr, R[t]);
        endif
    endif
    if PSTATE.EL == EL2 then
        AArch32.DTLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL3 then
        AArch32.DTLBI_ASID(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_NSH,
        TLBI_AllAttr, R[t]);
    endif

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DTLBIMVA, Data TLB Invalidate by VA

The DTLBIMVA characteristics are:

Purpose

Invalidate all cached copies of translation table entries from data TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
 - The entry is from a level of lookup above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to DTLBIMVA are UNDEFINED.

Attributes

DTLBIMVA is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA | | | | | | | | | | | | RES0 | | | | ASID | | | | | | | | | | | | | | | |

VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

Executing the DTLBIMVA instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
        && IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
            AArch32.DTLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBILevel_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.DTLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBILevel_Any, TLBI_AllAttr, R[t]);
        endif
    endif
elsif PSTATE.EL == EL2 then
    AArch32.DTLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH, TLBILevel_Any,
    TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.DTLBI_VA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_NSH,
    TLBILevel_Any, TLBI_AllAttr, R[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DVPRCTX, Data Value Prediction Restriction by Context

The DVPRCTX characteristics are:

Purpose

Data Value Prediction Restriction by Context applies to all Data Value Prediction Resources that predict execution based on information gathered within the target execution context or contexts.

Data value predictions determined by the actions of code in the target execution context or contexts appearing in program order before the instruction cannot exploitatively control speculative execution occurring after the instruction is complete and synchronized.

This instruction is guaranteed to be complete following a DSB that covers both read and write behavior on the same PE as executed the original restriction instruction, and a subsequent context synchronization event is required to ensure that the effect of the completion of the instructions is synchronized to the current execution.

Note

This instruction does not require the invalidation of prediction structures so long as the behavior described for completion of this instruction is met by the implementation.

On some implementations the instruction is likely to take a significant number of cycles to execute. This instruction is expected to be used very rarely, such as on the roll-over of an ASID or VMID, but should not be used on every context switch.

Configuration

This instruction is present only when AArch32 is supported and FEAT_SPECRES is implemented. Otherwise, direct accesses to DVPRCTX are UNDEFINED.

Attributes

DVPRCTX is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|----|------|----|----|----|----|----|----|----|------|----|----|----|-------|------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | GVMID | NS | EL | VMID | | | | | | | | RES0 | | | | GASID | ASID | | | | | | | | | | | |

Bits [31:28]

Reserved, RES0.

GVMID, bit [27]

Execution of this instruction applies to all VMIDs or a specified VMID.

| GVMID | Meaning |
|-------|---|
| 0b0 | Applies to specified VMID for an EL0 or EL1 target execution context. |
| 0b1 | Applies to all VMIDs for an EL0 or EL1 target execution context. |

For target execution contexts other than EL0 or EL1, this field is RES0.

If the instruction is executed at EL0 or EL1, this field has an Effective value of 0.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

NS, bit [26]

Security State.

| NS | Meaning |
|-----|-------------------|
| 0b0 | Secure state. |
| 0b1 | Non-secure state. |

If the instruction is executed in Non-secure state, this field has an Effective value of 1.

EL, bits [25:24]

Exception Level. Indicates the Exception level of the target execution context.

| EL | Meaning |
|------|---------|
| 0b00 | EL0. |
| 0b01 | EL1. |
| 0b10 | EL2. |
| 0b11 | EL3. |

If the instruction is executed at an Exception level lower than the specified level, this instruction is treated as a NOP.

VMID, bits [23:16]

Only applies when bit[27] is 0 and the target execution context is either:

- EL1.
- EL0 when ([HCR_EL2.E2H==0](#) or [HCR_EL2.TGE==0](#)) or EL2 is using AArch32 state.

Otherwise this field is RES0.

When the instruction is executed at EL1, this field is treated as the current VMID.

When the instruction is executed at EL0 and ([HCR_EL2.E2H==0](#) or [HCR_EL2.TGE==0](#) or [ELUsingAArch32\(EL2\)](#)), this field is treated as the current VMID.

When the instruction is executed at EL0 and ([HCR_EL2.E2H==1](#) and [HCR_EL2.TGE==1](#) and [!ELUsingAArch32\(EL2\)](#)), this field is ignored.

If EL2 is not implemented or not enabled for the target Security state, this field is RES0.

Bits [15:9]

Reserved, RES0.

GASID, bit [8]

Execution of this instruction applies to all ASIDs or a specified ASID.

| GASID | Meaning |
|-------|--|
| 0b0 | Applies to specified ASID for an EL0 target execution context. |
| 0b1 | Applies to all ASID for an EL0 target execution context. |

For target execution contexts other than EL0, this field is RES0.

If the instruction is executed at EL0, this field has an Effective value of 0.

ASID, bits [7:0]

Only applies for an EL0 target execution context and when bit[8] is 0.

Otherwise, this field is RES0.

When the instruction is executed at EL0, this field is treated as the current ASID.

Executing the DVPRCTX instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b0011 | 0b101 |

```

if PSTATE.EL == EL0 then
    if !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && SCTLR_EL1.EnRCTX ==
'0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        elsif ELUsingAArch32(EL1) && SCTLR.EnRCTX == '0' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T7 == '1'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGITR_EL2.DVPRCTX == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && SCTLR_EL2.EnRCTX ==
'0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch32.RestrictPrediction(R[t], RestrictType_DataValue);
    elsif PSTATE.EL == EL1 then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.NV == '1' then
            AArch64.SystemAccessTrap(EL2, 0x03);
        else
            AArch32.RestrictPrediction(R[t], RestrictType_DataValue);
    elsif PSTATE.EL == EL2 then
        AArch32.RestrictPrediction(R[t], RestrictType_DataValue);
    elsif PSTATE.EL == EL3 then
        AArch32.RestrictPrediction(R[t], RestrictType_DataValue);

```

ELR_hyp, Exception Link Register (Hyp mode)

The ELR_hyp characteristics are:

Purpose

When taking an exception to Hyp mode, holds the address to return to.

Configuration

AArch32 System register ELR_hyp bits [31:0] are architecturally mapped to AArch64 System register [ELR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ELR_hyp are UNDEFINED.

Attributes

ELR_hyp is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Return address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Return address.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ELR_hyp

ELR_hyp is accessible only at Hyp mode and Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

MRS{<c>}{<q>} <Rd>, ELR_hyp

| R | M | M1 |
|-----|-----|--------|
| 0b0 | 0b1 | 0b1110 |

MSR{<c>}{<q>} ELR_hyp, <Rn>

| R | M | M1 |
|-----|-----|--------|
| 0b0 | 0b1 | 0b1110 |

ERRIDR, Error Record ID Register

The ERRIDR characteristics are:

Purpose

Defines the highest numbered index of the error records that can be accessed through the Error Record System registers.

Configuration

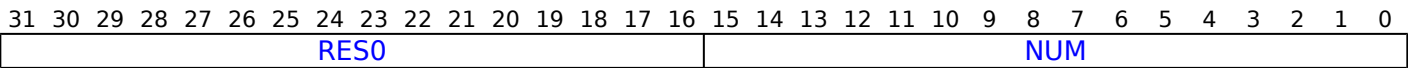
AArch32 System register ERRIDR bits [31:0] are architecturally mapped to AArch64 System register [ERRIDR_EL1\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERRIDR are UNDEFINED.

Attributes

ERRIDR is a 32-bit register.

Field descriptions



Bits [31:16]

Reserved, RES0.

NUM, bits [15:0]

Highest numbered index of the records that can be accessed through the Error Record System registers plus one. Zero indicates that no records can be accessed through the Error Record System registers.

Each implemented record is owned by a node. A node might own multiple records.

Accessing ERRIDR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERRIDR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERRIDR;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERRIDR;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRSELR, Error Record Select Register

The ERRSELR characteristics are:

Purpose

Selects an error record to be accessed through the Error Record System registers.

Configuration

AArch32 System register ERRSELR bits [31:0] are architecturally mapped to AArch64 System register [ERRSELR_EL1\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERRSELR are UNDEFINED.

If [ERRIDR](#) indicates that zero error records are implemented, then it is IMPLEMENTATION DEFINED whether ERRSELR is UNDEFINED or RES0.

Attributes

ERRSELR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | SEL | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

SEL, bits [15:0]

Selects the error record accessed through the ERX registers.

For example, if ERRSELR.SEL is 0x0004, then direct reads and writes of [ERXSTATUS](#) access ERR4STATUS.

If ERRSELR.SEL is greater than or equal to [ERRIDR.NUM](#), then all of the following apply:

- The value read back from ERRSELR.SEL is UNKNOWN.
- One of the following occurs:
 - An UNKNOWN error record is selected.
 - The ERX* registers are RAZ/WI.
 - ERX* register reads and writes are NOPs.
 - ERX* register reads and writes are UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ERRSELR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| | | | | |
|--------|------|-----|-----|------|
| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0011 | 0b001 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERRSELR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERRSELR;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERRSELR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERRSELR = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERRSELR = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERRSELR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXADDR, Selected Error Record Address Register

The ERXADDR characteristics are:

Purpose

Accesses bits [31:0] of [ERR<n>ADDR](#) for the error record <n> selected by [ERRSEL](#).SEL.

Configuration

AArch32 System register ERXADDR bits [31:0] are architecturally mapped to AArch64 System register [ERXADDR_EL1\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXADDR are UNDEFINED.

Attributes

ERXADDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Bits [31:0] of ERR<n>ADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

ERXADDR accesses bits [31:0] of [ERR<n>ADDR](#), where <n> is the value in [ERRSEL](#).SEL.

Accessing ERXADDR

If [ERRIDR](#).NUM is 0x0000 or [ERRSEL](#).SEL is greater than or equal to [ERRIDR](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXADDR is RAZ/WI.
- Direct reads and writes of ERXADDR are NOPs.
- Direct reads and writes of ERXADDR are UNDEFINED.

[ERR<n>ADDR](#) describes additional constraints that also apply when [ERR<n>ADDR](#) is accessed through ERXADDR.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXADDR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXADDR;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXADDR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXADDR = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXADDR = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXADDR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXADDR2, Selected Error Record Address Register 2

The ERXADDR2 characteristics are:

Purpose

Accesses bits [63:32] of [ERR<n>ADDR](#) for the error record <n> selected by [ERRSEL](#).SEL.

Configuration

AArch32 System register ERXADDR2 bits [31:0] are architecturally mapped to AArch64 System register [ERXADDR_EL1\[63:32\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXADDR2 are UNDEFINED.

Attributes

ERXADDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Bits [63:32] of ERR<n>ADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

ERXADDR2 accesses bits [63:32] of [ERR<n>ADDR](#), where <n> is the value in [ERRSEL](#).SEL.

Accessing ERXADDR2

If [ERRIDR](#).NUM is 0x0000 or [ERRSEL](#).SEL is greater than or equal to [ERRIDR](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXADDR2 is RAZ/WI.
- Direct reads and writes of ERXADDR2 are NOPs.
- Direct reads and writes of ERXADDR2 are UNDEFINED.

[ERR<n>ADDR](#) describes additional constraints that also apply when [ERR<n>ADDR](#) is accessed through ERXADDR2.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXADDR2;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXADDR2;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXADDR2;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXADDR2 = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXADDR2 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXADDR2 = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXCTLR, Selected Error Record Control Register

The ERXCTRLR characteristics are:

Purpose

Accesses bits [31:0] of [ERR<n>CTRL](#) for the error record <n> selected by [ERRSEL](#).SEL.

Configuration

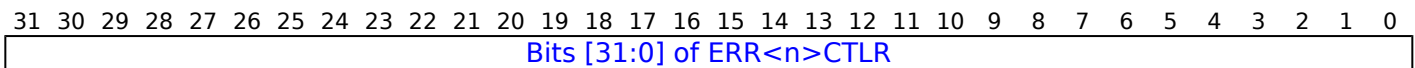
AArch32 System register ERXCTLR bits [31:0] are architecturally mapped to AArch64 System register [ERXCTLR_EL1\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXCTLR are UNDEFINED.

Attributes

ERXCTLR is a 32-bit register.

Field descriptions



Bits [31:0]

ERXCTLR accesses bits [31:0] of [ERR<n>CTLR](#), where <n> is the value in [ERRSEL](#).SEL.

Accessing ERXCTLR

If [ERRIDR.NUM](#) is 0x0000 or [ERRSEL.SEL](#) is greater than or equal to [ERRIDR.NUM](#), then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXCTL is RAZ/WI.
- Direct reads and writes of ERXCTL are NOPs.
- Direct reads and writes of ERXCTL are UNDEFINED.

If [ERRSEL](#).SEL is not the index of the first error record owned by a node, then [ERR<n>CTL](#)R[31:0] is not present, meaning reads and writes of [ERXCTL](#)R are RES0.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXCTLR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXCTLR;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXCTLR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXCTLR = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXCTLR = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXCTLR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXCTLR2, Selected Error Record Control Register 2

The ERXCTLR2 characteristics are:

Purpose

Accesses bits [63:32] of [ERR<n>CTLR](#) for the error record <n> selected by [ERRSELR](#).SEL.

Configuration

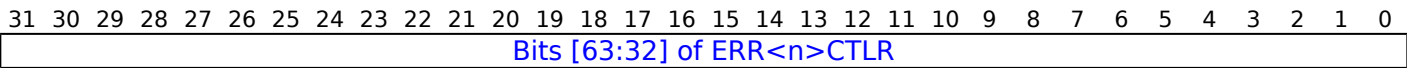
AArch32 System register ERXCTLR2 bits [31:0] are architecturally mapped to AArch64 System register [ERXCTLR_EL1\[63:32\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXCTLR2 are UNDEFINED.

Attributes

ERXCTLR2 is a 32-bit register.

Field descriptions



Bits [31:0]

ERXCTLR2 accesses bits [63:32] of [ERR<n>CTLR](#), where <n> is the value in [ERRSELR](#).SEL.

Accessing ERXCTLR2

If [ERRIDR](#).NUM is 0x0000 or [ERRSELR](#).SEL is greater than or equal to [ERRIDR](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXCTLR2 is RAZ/WI.
- Direct reads and writes of ERXCTLR2 are NOPs.
- Direct reads and writes of ERXCTLR2 are UNDEFINED.

If [ERRSELR](#).SEL is not the index of the first error record owned by a node, then [ERR<n>CTLR](#)[63:32] is not present, meaning reads and writes of ERXCTLR2 are RES0.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXCTLR2;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXCTLR2;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXCTLR2;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXCTLR2 = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ERXCTLR2 = R[t];
elsif PSTATE.EL == EL3 then
    if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
        AArch32.TakeMonitorTrapException();
    else
        ERXCTLR2 = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXFR, Selected Error Record Feature Register

The ERXFR characteristics are:

Purpose

Accesses bits [31:0] of [ERR<n>FR](#) for the error record <n> selected by [ERRSELR.SEL](#).

Configuration

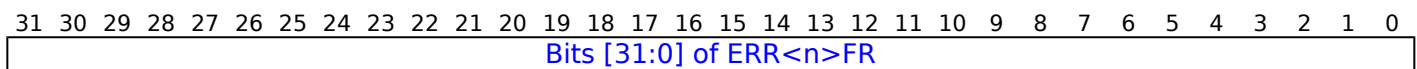
AArch32 System register ERXFR bits [31:0] are architecturally mapped to AArch64 System register [ERXFR_EL1\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXFR are UNDEFINED.

Attributes

ERXFR is a 32-bit register.

Field descriptions



Bits [31:0]

ERXFR accesses bits [31:0] of [ERR<n>FR](#), where <n> is the value in [ERRSELR.SEL](#).

Accessing ERXFR

If [ERRIDR.NUM](#) is 0x0000 or [ERRSELR.SEL](#) is greater than or equal to [ERRIDR.NUM](#), then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXFR is RAZ.
- Direct reads of ERXFR are NOPs.
- Direct reads of ERXFR are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXFR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXFR;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXFR;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXFR2, Selected Error Record Feature Register 2

The ERXFR2 characteristics are:

Purpose

Accesses bits [63:32] of [ERR<n>FR](#) for the error record <n> selected by [ERRSELR](#).SEL.

Configuration

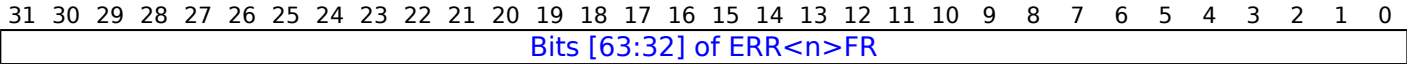
AArch32 System register ERXFR2 bits [31:0] are architecturally mapped to AArch64 System register [ERXFR_EL1\[63:32\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXFR2 are UNDEFINED.

Attributes

ERXFR2 is a 32-bit register.

Field descriptions



Bits [31:0]

ERXFR2 accesses bits [63:32] of [ERR<n>FR](#), where <n> is the value in [ERRSELR](#).SEL.

Accessing ERXFR2

If [ERRIDR](#).NUM is 0x0000 or [ERRSELR](#).SEL is greater than or equal to [ERRIDR](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXFR2 is RAZ.
- Direct reads of ERXFR2 are NOPs.
- Direct reads of ERXFR2 are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXFR2;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXFR2;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXFR2;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC0, Selected Error Record Miscellaneous Register 0

The ERXMISC0 characteristics are:

Purpose

Accesses bits [31:0] of [ERR<n>MISC0](#) for the error record <n> selected by [ERRSELR](#).SEL.

Configuration

AArch32 System register ERXMISC0 bits [31:0] are architecturally mapped to AArch64 System register [ERXMISC0_EL1\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXMISC0 are UNDEFINED.

Attributes

ERXMISC0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Bits [31:0] of ERR<n>MISC0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

ERXMISC0 accesses bits [31:0] of [ERR<n>MISC0](#), where <n> is the value in [ERRSELR](#).SEL.

Accessing ERXMISC0

If [ERRIDR](#).NUM is 0x0000 or [ERRSELR](#).SEL is greater than or equal to [ERRIDR](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC0 is RAZ/WI.
- Direct reads and writes of ERXMISC0 are NOPs.
- Direct reads and writes of ERXMISC0 are UNDEFINED.

[ERR<n>MISC0](#) describes additional constraints that also apply when [ERR<n>MISC0](#) is accessed through ERXMISC0.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC0;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXMISC0;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC0 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC0 = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXMISC0 = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC1, Selected Error Record Miscellaneous Register 1

The ERXMISC1 characteristics are:

Purpose

Accesses bits [63:32] of [ERR<n>MISC0](#) for the error record <n> selected by [ERRSELR](#).SEL.

Configuration

AArch32 System register ERXMISC1 bits [31:0] are architecturally mapped to AArch64 System register [ERXMISC0_EL1\[63:32\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXMISC1 are UNDEFINED.

Attributes

ERXMISC1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Bits [63:32] of ERR<n>MISC0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

ERXMISC1 accesses bits [63:32] of [ERR<n>MISC0](#), where <n> is the value in [ERRSELR](#).SEL.

Accessing ERXMISC1

If [ERRIDR](#).NUM is 0x0000 or [ERRSELR](#).SEL is greater than or equal to [ERRIDR](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC1 is RAZ/WI.
- Direct reads and writes of ERXMISC1 are NOPs.
- Direct reads and writes of ERXMISC1 are UNDEFINED.

[ERR<n>MISC0](#) describes additional constraints that also apply when [ERR<n>MISC0](#) is accessed through ERXMISC1.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC1;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXMISC1;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC1 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC1 = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXMISC1 = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC2, Selected Error Record Miscellaneous Register 2

The ERXMISC2 characteristics are:

Purpose

Accesses bits [31:0] of [ERR<n>MISC1](#) for the error record <n> selected by [ERRSELR](#).SEL.

Configuration

AArch32 System register ERXMISC2 bits [31:0] are architecturally mapped to AArch64 System register [ERXMISC1_EL1\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXMISC2 are UNDEFINED.

Attributes

ERXMISC2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Bits [31:0] of ERR<n>MISC1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

ERXMISC2 accesses bits [31:0] of [ERR<n>MISC1](#), where <n> is the value in [ERRSELR](#).SEL.

Accessing ERXMISC2

If [ERRIDR](#).NUM is 0x0000 or [ERRSELR](#).SEL is greater than or equal to [ERRIDR](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC2 is RAZ/WI.
- Direct reads and writes of ERXMISC2 are NOPs.
- Direct reads and writes of ERXMISC2 are UNDEFINED.

[ERR<n>MISC1](#) describes additional constraints that also apply when [ERR<n>MISC1](#) is accessed through ERXMISC2.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC2;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC2;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXMISC2;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC2 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC2 = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXMISC2 = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC3, Selected Error Record Miscellaneous Register 3

The ERXMISC3 characteristics are:

Purpose

Accesses bits [63:32] of [ERR<n>MISC1](#) for the error record <n> selected by [ERRSELR](#).SEL.

Configuration

AArch32 System register ERXMISC3 bits [31:0] are architecturally mapped to AArch64 System register [ERXMISC1_EL1\[63:32\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXMISC3 are UNDEFINED.

Attributes

ERXMISC3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Bits [63:32] of ERR<n>MISC1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

ERXMISC3 accesses bits [63:32] of [ERR<n>MISC1](#), where <n> is the value in [ERRSELR](#).SEL.

Accessing ERXMISC3

If [ERRIDR](#).NUM is 0x0000 or [ERRSELR](#).SEL is greater than or equal to [ERRIDR](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC3 is RAZ/WI.
- Direct reads and writes of ERXMISC3 are NOPs.
- Direct reads and writes of ERXMISC3 are UNDEFINED.

[ERR<n>MISC1](#) describes additional constraints that also apply when [ERR<n>MISC1](#) is accessed through ERXMISC3.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC3;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC3;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXMISC3;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC3 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC3 = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXMISC3 = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC4, Selected Error Record Miscellaneous Register 4

The ERXMISC4 characteristics are:

Purpose

Accesses bits [31:0] of [ERR<n>MISC2](#) for the error record <n> selected by [ERRSELR](#).SEL.

Configuration

AArch32 System register ERXMISC4 bits [31:0] are architecturally mapped to AArch64 System register [ERXMISC2_EL1\[31:0\]](#).

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXMISC4 are UNDEFINED.

Attributes

ERXMISC4 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Bits [31:0] of ERR<n>MISC2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

ERXMISC4 accesses bits [31:0] of [ERR<n>MISC2](#), where <n> is the value in [ERRSELR](#).SEL.

Accessing ERXMISC4

If [ERRIDR](#).NUM is 0x0000 or [ERRSELR](#).SEL is greater than or equal to [ERRIDR](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC4 is RAZ/WI.
- Direct reads and writes of ERXMISC4 are NOPs.
- Direct reads and writes of ERXMISC4 are UNDEFINED.

[ERR<n>MISC2](#) describes additional constraints that also apply when [ERR<n>MISC2](#) is accessed through ERXMISC4.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC4;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC4;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXMISC4;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC4 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC4 = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXMISC4 = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC5, Selected Error Record Miscellaneous Register 5

The ERXMISC5 characteristics are:

Purpose

Accesses bits [63:32] of [ERR<n>MISC2](#) for the error record <n> selected by [ERRSELR](#).SEL.

Configuration

AArch32 System register ERXMISC5 bits [31:0] are architecturally mapped to AArch64 System register [ERXMISC2_EL1\[63:32\]](#).

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXMISC5 are UNDEFINED.

Attributes

ERXMISC5 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Bits [63:32] of ERR<n>MISC2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

ERXMISC5 accesses bits [63:32] of [ERR<n>MISC2](#), where <n> is the value in [ERRSELR](#).SEL.

Accessing ERXMISC5

If [ERRIDR](#).NUM is 0x0000 or [ERRSELR](#).SEL is greater than or equal to [ERRIDR](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC5 is RAZ/WI.
- Direct reads and writes of ERXMISC5 are NOPs.
- Direct reads and writes of ERXMISC5 are UNDEFINED.

[ERR<n>MISC2](#) describes additional constraints that also apply when [ERR<n>MISC2](#) is accessed through ERXMISC5.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC5;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC5;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXMISC5;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC5 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC5 = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXMISC5 = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC6, Selected Error Record Miscellaneous Register 6

The ERXMISC6 characteristics are:

Purpose

Accesses bits [31:0] of [ERR<n>MISC3](#) for the error record <n> selected by [ERRSELR](#).SEL.

Configuration

AArch32 System register ERXMISC6 bits [31:0] are architecturally mapped to AArch64 System register [ERXMISC3_EL1\[31:0\]](#).

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXMISC6 are UNDEFINED.

Attributes

ERXMISC6 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Bits [31:0] of ERR<n>MISC3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

ERXMISC6 accesses bits [31:0] of [ERR<n>MISC3](#), where <n> is the value in [ERRSELR](#).SEL.

Accessing ERXMISC6

If [ERRIDR](#).NUM is 0x0000 or [ERRSELR](#).SEL is greater than or equal to [ERRIDR](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC6 is RAZ/WI.
- Direct reads and writes of ERXMISC6 are NOPs.
- Direct reads and writes of ERXMISC6 are UNDEFINED.

[ERR<n>MISC3](#) describes additional constraints that also apply when [ERR<n>MISC3](#) is accessed through ERXMISC6.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC6;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC6;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXMISC6;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC6 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC6 = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXMISC6 = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXMISC7, Selected Error Record Miscellaneous Register 7

The ERXMISC7 characteristics are:

Purpose

Accesses bits [63:32] of [ERR<n>MISC3](#) for the error record <n> selected by [ERRSELR](#).SEL.

Configuration

AArch32 System register ERXMISC7 bits [31:0] are architecturally mapped to AArch64 System register [ERXMISC3_EL1\[63:32\]](#).

This register is present only when FEAT_RASv1p1 is implemented. Otherwise, direct accesses to ERXMISC7 are UNDEFINED.

Attributes

ERXMISC7 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Bits [63:32] of ERR<n>MISC3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

ERXMISC7 accesses bits [63:32] of [ERR<n>MISC3](#), where <n> is the value in [ERRSELR](#).SEL.

Accessing ERXMISC7

If [ERRIDR](#).NUM is 0x0000 or [ERRSELR](#).SEL is greater than or equal to [ERRIDR](#).NUM, then one of the following occurs:

- An UNKNOWN error record is selected.
- ERXMISC7 is RAZ/WI.
- Direct reads and writes of ERXMISC7 are NOPs.
- Direct reads and writes of ERXMISC7 are UNDEFINED.

[ERR<n>MISC3](#) describes additional constraints that also apply when [ERR<n>MISC3](#) is accessed through ERXMISC7.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC7;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXMISC7;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXMISC7;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC7 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXMISC7 = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXMISC7 = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERXSTATUS, Selected Error Record Primary Status Register

The ERXSTATUS characteristics are:

Purpose

Accesses bits [31:0] of [ERR<n>STATUS](#) for the error record selected by [ERRSELR.SEL](#).

Configuration

AArch32 System register ERXSTATUS bits [31:0] are architecturally mapped to AArch64 System register [ERXSTATUS_EL1\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to ERXSTATUS are UNDEFINED.

Attributes

ERXSTATUS is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Bits [31:0] of ERR<n>STATUS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

ERXSTATUS accesses bits [31:0] of [ERR<n>STATUS](#), where n is the value in [ERRSELR.SEL](#).

Accessing ERXSTATUS

If [ERRIDR.NUM](#) == 0 or [ERRSELR.SEL](#) is set to a value greater than or equal to [ERRIDR.NUM](#), then one of the following occurs:

- An UNKNOWN record is selected.
- ERXSTATUS is RAZ/WI.
- Direct reads and writes of ERXSTATUS are NOPs.
- Direct reads and writes of ERXSTATUS are UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXSTATUS;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ERXSTATUS;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return ERXSTATUS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TERR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TERR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXSTATUS = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.TERR == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ERXSTATUS = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SCR.TERR == '1' then
            AArch32.TakeMonitorTrapException();
        else
            ERXSTATUS = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FCSEIDR, FCSE Process ID register

The FCSEIDR characteristics are:

Purpose

Identifies whether the Fast Context Switch Extension (FCSE) is implemented.

From Armv8, the FCSE is not implemented, so this register is RAZ/WI. Software can access this register to determine that the implementation does not include the FCSE.

Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to FCSEIDR are UNDEFINED.

Attributes

FCSEIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | RAZ/WI | | | | | | | | | | | | | | | |

Bits [31:0]

Reserved, RAZ/WI.

Accessing FCSEIDR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return FCSEIDR;
elsif PSTATE.EL == EL2 then
    return FCSEIDR;
elsif PSTATE.EL == EL3 then
    return FCSEIDR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b000 |
|--------|-------|--------|--------|-------|

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        FCSEIDR = R[t];
elsif PSTATE.EL == EL2 then
    FCSEIDR = R[t];
elsif PSTATE.EL == EL3 then
    FCSEIDR = R[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FPEXC, Floating-Point Exception Control register

The FPEXC characteristics are:

Purpose

Provides a global enable for the implemented Advanced SIMD and floating-point functionality, and reports floating-point status information.

Configuration

AArch32 System register FPEXC bits [31:0] are architecturally mapped to AArch64 System register [FPEXC32_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to FPEXC are UNDEFINED.

Implemented only if the implementation includes the Advanced SIMD and floating-point functionality.

Attributes

FPEXC is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|-----|------|----|-----|------|----|----|----|----|----|----|----|----|----|--------|----|----|-----|------|-----|-----|-----|-----|-----|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EX | EN | DEX | FP2V | VV | TFV | RES0 | | | | | | | | | | VECITR | | | IDF | RES0 | IXF | UFF | OFF | DZF | IOF | | | | | | |

EX, bit [31]

Exception bit. From Armv8, this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EN, bit [30]

Enables access to the Advanced SIMD and floating-point functionality from all Exception levels, except that setting this field to 0 does not disable the following:

- VMSR accesses to the [FPEXC](#) or [FPSID](#).
- VMRS accesses from the [FPEXC](#), [FPSID](#), [MVFR0](#), [MVFR1](#), or [MVFR2](#).

| EN | Meaning |
|-----|--|
| 0b0 | Accesses to the FPSCR , and any of the SIMD and floating-point registers Q0-Q15, including their views as D0-D31 registers or S0-S31 registers, are UNDEFINED at all Exception levels. |
| 0b1 | This control permits access to the Advanced SIMD and floating-point functionality at all Exception levels. |

Execution of Advanced SIMD and floating-point instructions in AArch32 state can be disabled or trapped by the following controls:

- [CPACR](#).cp10, or, if executing at EL0, [CPACR_EL1](#).FPEN.
- FPEXC.EN.
- If executing in Non-secure state:
 - [HCPTR](#).TCP10, or if EL2 is using AArch64, [CPTR_EL2](#).TFP.
 - [NSACR](#).cp10, or if EL3 is using AArch64, [CPTR_EL3](#).TFP.
- For Advanced SIMD instructions only:
 - CPACR.ASEDIS.

- If executing in Non-secure state, [HCPTR.TASE](#) and [NSACR.NSTRCDIS](#).

See the descriptions of the controls for more information.

Note

When executing at EL0 using AArch32:

- If EL1 is using AArch64, then the Effective value of FPEXC.EN is 1. This includes when EL2 is using AArch64 and is enabled in the current Security state, [HCR_EL2.TGE](#) is 1, and the Effective value of [HCR_EL2.RW](#) is 1.
 - If EL2 is using AArch64 and is enabled in the current Security state, [HCR_EL2.TGE](#) is 1, and the Effective value of [HCR_EL2.RW](#) is 0, then it is IMPLEMENTATION DEFINED whether the Effective value of FPEXC.EN is 1 or the value written to FPEXC.EN. However, Arm deprecates using the value of FPEXC.EN to determine behavior.
-

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

DEX, bit [29]

Defined synchronous exception on floating-point execution.

This field identifies whether a synchronous exception generated by the attempted execution of an instruction was generated by an unallocated encoding. The instruction must be in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr() returning TRUE. This field also indicates whether the FPEXC.TFV field is valid.

The meaning of this bit is:

| DEX | Meaning |
|-----|---|
| 0b0 | The exception was generated by the attempted execution of an unallocated instruction in the encoding space that is identified by the pseudocode function ExecutingCP10or11Instr(). If FPEXC.TFV is RW then it is invalid and UNKNOWN. If FPEXC.{IDE, IXF, UFF, OFF, DZF, IOF} are RW then they are invalid and UNKNOWN. |
| 0b1 | The exception was generated during the execution of an allocated encoding. FPEXC.TFV is valid and indicates the cause of the exception. |

On an exception that sets this bit to 1 the exception-handling routine must clear this bit to 0.

On an implementation that both does not support trapping of floating-point exceptions and implements the [FPSCR](#).{Stride, Len} fields as RAZ, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FP2V, bit [28]

FPINST2 instruction valid bit. From Armv8, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VV, bit [27]

VECITR valid bit. From Armv8, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TFV, bit [26]

Trapped Fault Valid bit. Valid only when the value of FPEXC.DEX is 1. When valid, it indicates the cause of the exception and therefore whether the FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} bits are valid.

| TFV | Meaning |
|-----|---|
| 0b0 | The exception was caused by the execution of a floating-point VABS, VADD, VDIV, VFMA, VFMS, VFNMA, VFNMS, VMLA, VMLS, VMOV, VMUL, VNEG, VNMLA, VNMLS, VNMUL, VSQRT, or VSUB instruction when one or both of FPSCR .{Stride, Len} was non-zero. If the FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} bits are RW then they are invalid and UNKNOWN. |
| 0b1 | FPEXC.{IDF, IXF, UFF, OFF, DZF, IOF} indicate the presence of trapped floating-point exceptions that had occurred at the time of the exception. Bits are set for all trapped exceptions that had occurred at the time of the exception. |

This bit returns a status value and ignores writes.

When the value of FPEXC.DEX is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

On an implementation that supports the trapping of floating-point exceptions and implements [FPSCR](#).{Stride, Len} as RAZ, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [25:11]

Reserved, RES0.

VECITR, bits [10:8]

Vector iteration count. From Armv8, this field is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IDF, bit [7]

Input Denormal trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Input Denormal exception occurred while [FPSCR](#).IDE was 1:

| IDF | Meaning |
|-----|--|
| 0b0 | Input Denormal exception has not occurred. |
| 0b1 | Input Denormal exception has occurred. |

Input Denormal exceptions can occur only when [FPSCR](#).FZ is 1.

Note

A half-precision floating-point value that is flushed to zero because the value of [FPSCR](#).FZ16 is 1 does not generate an Input Denormal exception.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXF, bit [4]

Inexact trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Inexact exception occurred while [FPSCR.IXE](#) was 1:

| IXF | Meaning |
|-----|-------------------------------------|
| 0b0 | Inexact exception has not occurred. |
| 0b1 | Inexact exception has occurred. |

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFF, bit [3]

Underflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Underflow exception occurred while [FPSCR.UFE](#) was 1:

| UFF | Meaning |
|-----|---------------------------------------|
| 0b0 | Underflow exception has not occurred. |
| 0b1 | Underflow exception has occurred. |

Underflow trapped exceptions can occur:

- On half-precision data-processing instructions only when [FPSCR.FZ16](#) is 0.
- Otherwise only when [FPSCR.FZ](#) is 0.

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFF, bit [2]

Overflow trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Overflow exception occurred while [FPSCR.OFE](#) was 1:

| OFF | Meaning |
|-----|--------------------------------------|
| 0b0 | Overflow exception has not occurred. |
| 0b1 | Overflow exception has occurred. |

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZF, bit [1]

Divide by Zero trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether a Divide by Zero exception occurred while [FPSCR.DZE](#) was 1:

| DZF | Meaning |
|------------|--|
| 0b0 | Divide by Zero exception has not occurred. |
| 0b1 | Divide by Zero exception has occurred. |

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOF, bit [0]

Invalid Operation trapped exception bit. Valid only when the value of FPEXC.TFV is 1. When valid, it indicates whether an Invalid Operation exception occurred while [FPSCR.IOE](#) was 1:

| IOF | Meaning |
|------------|---|
| 0b0 | Invalid Operation exception has not occurred. |
| 0b1 | Invalid Operation exception has occurred. |

This bit must be cleared to 0 by the exception-handling routine.

When the value of FPEXC.TFV is 0 and this bit is RW, this bit is invalid and UNKNOWN.

On an implementation that does not support the trapping of floating-point exceptions this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing FPEXC

Accesses to this register use the following encodings in the System register encoding space:

VMRS{<c>}{<q>} <Rt>, <spec_reg>

| reg |
|------------|
| 0b1000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
        else
            return FPEXC;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            UNDEFINED;
        elsif EL2Enabled() && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
HCPTR.TCP10 == '1') then
            AArch32.TakeHypTrapException(0x00);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x07);
            else
                return FPEXC;
    elsif PSTATE.EL == EL3 then
        if CPACR.cp10 == '00' then
            UNDEFINED;
        else
            return FPEXC;

```

VMSR{<c>}{<q>} <spec_reg>, <Rt>

| reg |
|--------|
| 0b1000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
        else
            FPExC = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif EL2Enabled() && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x00);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
        else
            FPExC = R[t];
elsif PSTATE.EL == EL3 then
    if CPACR.cp10 == '00' then
        UNDEFINED;
    else
        FPExC = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FPSCR, Floating-Point Status and Control Register

The FPSCR characteristics are:

Purpose

Provides floating-point system status information and control.

Configuration

AArch32 System register FPSCR bits [31:27] are architecturally mapped to AArch64 System register [FPSR\[31:27\]](#).

AArch32 System register FPSCR bit [7] is architecturally mapped to AArch64 System register [FPSR\[7\]](#).

AArch32 System register FPSCR bits [4:0] are architecturally mapped to AArch64 System register [FPSR\[4:0\]](#).

AArch32 System register FPSCR bits [26:15] are architecturally mapped to AArch64 System register [FPSR\[26:15\]](#).

AArch32 System register FPSCR bits [12:8] are architecturally mapped to AArch64 System register [FPSR\[12:8\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to FPSCR are UNDEFINED.

It is IMPLEMENTATION DEFINED whether the Len and Stride fields can be programmed to non-zero values, which will cause some AArch32 floating-point instruction encodings to be UNDEFINED, or whether these fields are RAZ.

Implemented only if the implementation includes the Advanced SIMD and floating-point functionality.

Attributes

FPSCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|------|--------|----|----|-----|----|------|-----|-----|----|----|-----|-----|----|-------|-----|-----|----|----|-----|-----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
| N | Z | C | V | Q | A | H | P | D | N | F | Z | R | Mode | Stride | FZ | 16 | Len | ID | RES0 | IXE | UFE | OF | ED | DZE | IOE | ID | CRES0 | IXC | UFC | OF | CD | DZC | IOC |

N, bit [31]

Negative condition flag. This is updated by floating-point comparison operations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero condition flag. This is updated by floating-point comparison operations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry condition flag. This is updated by floating-point comparison operations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow condition flag. This is updated by floating-point comparison operations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

QC, bit [27]

Cumulative saturation bit, Advanced SIMD only. This bit is set to 1 to indicate that an Advanced SIMD integer operation has saturated since 0 was last written to this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AHP, bit [26]

Alternative half-precision control bit:

| AHP | Meaning |
|------------|---|
| 0b0 | IEEE half-precision format selected. |
| 0b1 | Alternative half-precision format selected. |

This bit is used only for conversions between half-precision floating-point and other floating-point formats.

The data-processing instructions added as part of the FEAT_FP16 extension always use the IEEE half-precision format, and ignore the value of this bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DN, bit [25]

Default NaN mode control bit:

| DN | Meaning |
|-----------|---|
| 0b0 | NaN operands propagate through to the output of a floating-point operation. |
| 0b1 | Any operation involving one or more NaNs returns the Default NaN. |

The value of this bit controls only scalar floating-point arithmetic. Advanced SIMD arithmetic always uses the Default NaN setting, regardless of the value of the DN bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FZ, bit [24]

Flush-to-zero mode control bit:

| FZ | Meaning |
|-----------|---|
| 0b0 | Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard. |
| 0b1 | Flush-to-zero mode enabled. |

The value of this bit controls only scalar floating-point arithmetic. Advanced SIMD arithmetic always uses the Flush-to-zero setting, regardless of the value of the FZ bit.

This bit has no effect on half-precision calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RMode, bits [23:22]

Rounding Mode control field. The encoding of this field is:

| RMode | Meaning |
|-------|---|
| 0b00 | Round to Nearest (RN) mode. |
| 0b01 | Round towards Plus Infinity (RP) mode. |
| 0b10 | Round towards Minus Infinity (RM) mode. |
| 0b11 | Round towards Zero (RZ) mode. |

The specified rounding mode is used by almost all scalar floating-point instructions. Advanced SIMD arithmetic always uses the Round to Nearest setting, regardless of the value of the RMode bits.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Stride, bits [21:20]

It is IMPLEMENTATION DEFINED whether this field is RW or RAZ.

If this field is RW and is set to a value other than zero, some floating-point instruction encodings are UNDEFINED. The instruction pseudocode identifies these instructions.

Arm strongly recommends that software never sets this field to a value other than zero.

The value of this field is ignored when processing Advanced SIMD instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FZ16, bit [19]

When FEAT_FP16 is implemented:

Flush-to-zero mode control bit on half-precision data-processing instructions:

| FZ16 | Meaning |
|------|---|
| 0b0 | Flush-to-zero mode disabled. Behavior of the floating-point system is fully compliant with the IEEE 754 standard. |
| 0b1 | Flush-to-zero mode enabled. |

The value of this bit applies to both scalar and Advanced SIMD floating-point half-precision calculations.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Len, bits [18:16]

It is IMPLEMENTATION DEFINED whether this field is RW or RAZ.

If this field is RW and is set to a value other than zero, some floating-point instruction encodings are UNDEFINED. The instruction pseudocode identifies these instructions.

Arm strongly recommends that software never sets this field to a value other than zero.

The value of this field is ignored when processing Advanced SIMD instructions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IDE, bit [15]

Input Denormal floating-point exception trap enable.

| IDE | Meaning |
|-----|--|
| 0b0 | Untrapped exception handling selected. If the floating-point exception occurs, the IDC bit is set to 1. |
| 0b1 | Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the IDC bit. |

This bit is RW only if the implementation supports the trapping of floating-point exceptions. In an implementation that does not support floating-point exception trapping, this bit is RAZ/WI.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [14:13]

Reserved, RES0.

IXE, bit [12]

Inexact floating-point exception trap enable.

| IXE | Meaning |
|-----|--|
| 0b0 | Untrapped exception handling selected. If the floating-point exception occurs, the IXC bit is set to 1. |
| 0b1 | Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the IXC bit. |

This bit is RW only if the implementation supports the trapping of floating-point exceptions. In an implementation that does not support floating-point exception trapping, this bit is RAZ/WI.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFE, bit [11]

Underflow floating-point exception trap enable.

| UFE | Meaning |
|-----|---|
| 0b0 | Untrapped exception handling selected. If the floating-point exception occurs, the UFC bit is set to 1. |
| 0b1 | Trapped exception handling selected. If the floating-point exception occurs and Flush-to-zero is not enabled, the PE does not update the UFC bit. |

This bit is RW only if the implementation supports the trapping of floating-point exceptions. In an implementation that does not support floating-point exception trapping, this bit is RAZ/WI.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFE, bit [10]

Overflow floating-point exception trap enable.

| OFE | Meaning |
|-----|--|
| 0b0 | Untrapped exception handling selected. If the floating-point exception occurs, the OFC bit is set to 1. |
| 0b1 | Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the OFC bit. |

This bit is RW only if the implementation supports the trapping of floating-point exceptions. In an implementation that does not support floating-point exception trapping, this bit is RAZ/WI.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZE, bit [9]

Divide by Zero floating-point exception trap enable.

| DZE | Meaning |
|-----|--|
| 0b0 | Untrapped exception handling selected. If the floating-point exception occurs, the DZC bit is set to 1. |
| 0b1 | Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the DZC bit. |

This bit is RW only if the implementation supports the trapping of floating-point exceptions. In an implementation that does not support floating-point exception trapping, this bit is RAZ/WI.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOE, bit [8]

Invalid Operation floating-point exception trap enable.

| IOE | Meaning |
|-----|--|
| 0b0 | Untrapped exception handling selected. If the floating-point exception occurs, the IOC bit is set to 1. |
| 0b1 | Trapped exception handling selected. If the floating-point exception occurs, the PE does not update the IOC bit. |

This bit is RW only if the implementation supports the trapping of floating-point exceptions. In an implementation that does not support floating-point exception trapping, this bit is RAZ/WI.

When this bit is RW, it applies only to floating-point operations. Advanced SIMD operations always use untrapped floating-point exception handling in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IDC, bit [7]

Input Denormal cumulative floating-point exception bit. This bit is set to 1 to indicate that the Input Denormal floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the IDE bit.

Advanced SIMD instructions set this bit if the Input Denormal floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the IDE bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IXC, bit [4]

Inexact cumulative floating-point exception bit. This bit is set to 1 to indicate that the Inexact floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the IXE bit.

Advanced SIMD instructions set this bit if the Inexact floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the IXE bit.

The criteria for the Inexact floating-point exception to occur are different in Flush-to-zero mode. For more information, see 'Flush-to-zero'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UFC, bit [3]

Underflow cumulative floating-point exception bit. This bit is set to 1 to indicate that the Underflow floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the UFE bit.

Advanced SIMD instructions set this bit if the Underflow floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, if FPSCR.UFE is 0 or if Flush-to-zero is enabled.

The criteria for the Underflow floating-point exception to occur are different in Flush-to-zero mode. For more information, see 'Flush-to-zero'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

OFC, bit [2]

Overflow cumulative floating-point exception bit. This bit is set to 1 to indicate that the Overflow floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the OFE bit.

Advanced SIMD instructions set this bit if the Overflow floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the OFE bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DZC, bit [1]

Divide by Zero cumulative floating-point exception bit. This bit is set to 1 to indicate that the Divide by Zero floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the DZE bit.

Advanced SIMD instructions set this bit if the Divide by Zero floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the DZE bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IOC, bit [0]

Invalid Operation cumulative floating-point exception bit. This bit is set to 1 to indicate that the Invalid Operation floating-point exception has occurred since 0 was last written to this bit.

How VFP instructions update this bit depends on the value of the IOE bit.

Advanced SIMD instructions set this bit if the Invalid Operation floating-point exception occurs in one or more of the floating-point calculations performed by the instruction, regardless of the value of the IOE bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing FPSCR

Accesses to this register use the following encodings in the System register encoding space:

```
VMRS{<c>}{<q>} <Rt>, <spec_reg>
```

| reg |
|--------|
| 0b0001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CPACR_EL1.FPEN
!= '11' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x00);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x07);
        elsif ELUsingAArch32(EL1) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
CPACR.cp10 == '0x') then
            UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.FPEN !=
'11' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x07);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x07);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x07);
        elsif EL2Enabled() && ELUsingAArch32(EL1) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
            AArch32.TakeHypTrapException(0x08);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x07);
            else
                return FPSCR;
        elsif PSTATE.EL == EL1 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
                UNDEFINED;
            elsif CPACR_EL1.FPEN == 'x0' then
                AArch64.AArch32SystemAccessTrap(EL1, 0x07);
            elsif (ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || CPACR.cp10 == '00' then
                UNDEFINED;
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x07);
            elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x07);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
                AArch32.TakeHypTrapException(0x08);
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x07);
                else
                    return FPSCR;
            elsif PSTATE.EL == EL2 then
                if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
                    UNDEFINED;
                elsif EL2Enabled() && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
HCPTR.TCP10 == '1') then
                    AArch32.TakeHypTrapException(0x00);
                elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch64.AArch32SystemAccessTrap(EL3, 0x07);
                    else
                        return FPSCR;
            elsif PSTATE.EL == EL3 then
                if CPACR.cp10 == '00' then
                    UNDEFINED;
                else

```

```
return FPSCR;
```

VMSR{<c>}{<q>} <spec_reg>, <Rt>

| reg |
|--------|
| 0b0001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && !(EL2Enabled() && HCR_EL2.<E2H,TGE> == '11') && CPACR_EL1.FPEN
!= '11' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x00);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x07);
        elsif ELUsingAArch32(EL1) && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
CPACR.cp10 == '0x') then
            UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> == '11' && CPTR_EL2.FPEN !=
'11' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x07);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x07);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x07);
        elsif EL2Enabled() && ELUsingAArch32(EL1) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
            AArch32.TakeHypTrapException(0x08);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x07);
            else
                FPSCR = R[t];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            UNDEFINED;
        elsif CPACR_EL1.FPEN == 'x0' then
            AArch64.AArch32SystemAccessTrap(EL1, 0x07);
        elsif (ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || CPACR.cp10 == '00' then
            UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x07);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x07);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
            AArch32.TakeHypTrapException(0x08);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x07);
            else
                FPSCR = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            UNDEFINED;
        elsif EL2Enabled() && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
HCPTR.TCP10 == '1') then
            AArch32.TakeHypTrapException(0x00);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x07);
            else
                FPSCR = R[t];
    elsif PSTATE.EL == EL3 then
        if CPACR.cp10 == '00' then
            UNDEFINED;
        else

```

```
FPSCR = R[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

FPSID, Floating-Point System ID register

The FPSID characteristics are:

Purpose

Provides top-level information about the floating-point implementation.
This register largely duplicates information held in the [MIDR](#). Arm deprecates use of it.

Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to FPSID are UNDEFINED.
Implemented only if the implementation includes the Advanced SIMD and floating-point functionality.

Attributes

FPSID is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|-----------------|----|----|----|----|----|----|---------|----|----|----|----|----|---------|---|---|---|----------|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Implementer | | | | | | | | SW | Subarchitecture | | | | | | | PartNum | | | | | | Variant | | | | Revision | | | | | |

Implementer, bits [31:24]

Implementer codes are the same as those used for the [MIDR](#).
For an implementation by Arm this field is 0x41, the ASCII code for A.
This field has an IMPLEMENTATION DEFINED value.
Access to this field is **RO**.

SW, bit [23]

Software bit. Defined values are:

| SW | Meaning |
|-----|---|
| 0b0 | The implementation provides a hardware implementation of the floating-point instructions. |
| 0b1 | The implementation supports only software emulation of the floating-point instructions. |

In Armv8-A, the only permitted value is 0b0.
Access to this field is **RO**.

Subarchitecture, bits [22:16]

Subarchitecture version number. For an implementation by Arm, defined values are:

| Subarchitecture | Meaning |
|-----------------|--|
| 0b0000000 | VFPv1 architecture with an IMPLEMENTATION DEFINED subarchitecture. |
| 0b0000001 | VFPv2 architecture with Common VFP subarchitecture v1. |
| 0b0000010 | VFPv3 architecture, or later, with Common VFP subarchitecture v2. The VFP architecture version is indicated by the MVFR0 and MVFR1 registers. |
| 0b0000011 | VFPv3 architecture, or later, with Null subarchitecture. The entire floating-point implementation is in hardware, and no software support code is required. The VFP architecture version is indicated by the MVFR0 and MVFR1 registers. This value can be used only by an implementation that does not support the trap enable bits in the FPSCR . |
| 0b0000100 | VFPv3 architecture, or later, with Common VFP subarchitecture v3, and support for trap enable bits in FPSCR . The VFP architecture version is indicated by the MVFR0 and MVFR1 registers. |

For a subarchitecture designed by Arm the most significant bit of this field, register bit[22], is 0. Values with a most significant bit of 0 that are not listed here are reserved.

When the subarchitecture designer is not Arm, the most significant bit of this field, register bit[22], must be 1. Each implementer must maintain its own list of subarchitectures it has designed, starting at subarchitecture version number 0x40.

In Armv8-A, the permitted values are 0b0000011 and 0b0000100.

Access to this field is **RO**.

PartNum, bits [15:8]

Part Number for the floating-point implementation, assigned by the implementer.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Variant, bits [7:4]

Variant number. Typically, this field distinguishes between different production variants of a single product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [3:0]

Revision number for the floating-point implementation.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing FPSID

Accesses to this register use the following encodings in the System register encoding space:

VMRS{<c>}{<q>} <Rt>, <spec_reg>

| reg |
|--------|
| 0b0000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x08);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID0 == '1' then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
        else
            return FPSID;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            UNDEFINED;
        elsif EL2Enabled() && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
HCPTR.TCP10 == '1') then
            AArch32.TakeHypTrapException(0x00);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x07);
            else
                return FPSID;
    elsif PSTATE.EL == EL3 then
        if CPACR.cp10 == '00' then
            UNDEFINED;
        else
            return FPSID;

```

VMSR{<c>}{<q>} <spec_reg>, <Rt>

| reg |
|--------|
| 0b0000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x08);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR_EL2.TID0 == '1' then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
        else
            //no operation
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            UNDEFINED;
        elsif EL2Enabled() && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
HCPTR.TCP10 == '1') then
            AArch32.TakeHypTrapException(0x00);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x07);
            else
                //no operation
    elsif PSTATE.EL == EL3 then
        if CPACR.cp10 == '00' then
            UNDEFINED;
        else
            //no operation

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HACR, Hyp Auxiliary Configuration Register

The HACR characteristics are:

Purpose

Controls trapping to Hyp mode of IMPLEMENTATION DEFINED aspects of Non-secure EL1 or EL0 operation.

Configuration

AArch32 System register HACR bits [31:0] are architecturally mapped to AArch64 System register [HACR_EL2\[31:0\]](#).

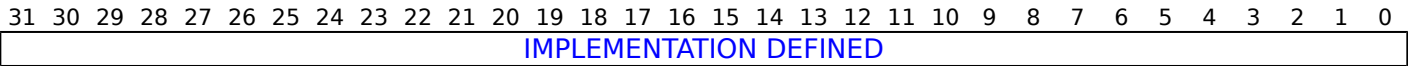
This register is present only when AArch32 is supported. Otherwise, direct accesses to HACR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HACR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HACR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HACR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HACR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HACR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HACR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HACTLR, Hyp Auxiliary Control Register

The HACTLR characteristics are:

Purpose

Controls IMPLEMENTATION DEFINED features of Hyp mode operation.

Configuration

AArch32 System register HACTLR bits [31:0] are architecturally mapped to AArch64 System register [ACTLR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HACTLR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HACTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HACTLR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HACTLR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HACTLR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HACTLR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HACTLR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HACTLR2, Hyp Auxiliary Control Register 2

The HACTLR2 characteristics are:

Purpose

Provides additional space to the HACTLR register to hold IMPLEMENTATION DEFINED trap functionality.

Configuration

AArch32 System register HACTLR2 bits [31:0] are architecturally mapped to AArch64 System register [ACTLR_EL2\[63:32\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HACTLR2 are UNDEFINED.

In Armv8.0 and Armv8.1, it is IMPLEMENTATION DEFINED whether this register is implemented, or whether it causes UNDEFINED exceptions when accessed. The implementation of this register can be detected by examining [ID_MMFR4.AC2](#).

From Armv8.2 this register must be implemented.

Attributes

HACTLR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HACTLR2

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HACTLR2;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HACTLR2;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HACTLR2 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HACTLR2 = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HADFSR, Hyp Auxiliary Data Fault Status Register

The HADFSR characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED syndrome information for Data Abort exceptions taken to Hyp mode.

Configuration

AArch32 System register HADFSR bits [31:0] are architecturally mapped to AArch64 System register [AFSR0_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HADFSR are UNDEFINED.

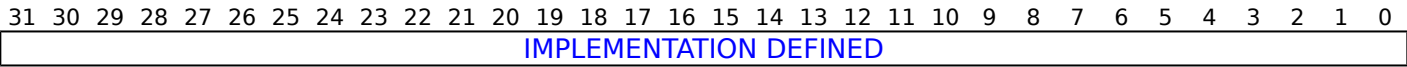
This is an optional register. An implementation that does not require this register can implement it as RES0.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HADFSR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HADFSR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0101 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HADFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HADFSR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0101 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HADFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HADFSR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HAIFSR, Hyp Auxiliary Instruction Fault Status Register

The HAIFSR characteristics are:

Purpose

Provides additional IMPLEMENTATION DEFINED syndrome information for Prefetch Abort exceptions taken to Hyp mode.

Configuration

AArch32 System register HAIFSR bits [31:0] are architecturally mapped to AArch64 System register [AFSR1_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HAIFSR are UNDEFINED.

This is an optional register. An implementation that does not require this register can implement it as RES0.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HAIFSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HAIFSR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0101 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HAIFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HAIFSR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0101 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HAIFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HAIFSR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HAMAIRO, Hyp Auxiliary Memory Attribute Indirection Register 0

The HAMAIRO characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory attribute encodings defined by [HMAIRO](#). These IMPLEMENTATION DEFINED attributes can only provide additional qualifiers for the memory attribute encodings, and cannot change the memory attributes defined in [HMAIRO](#).

Configuration

AArch32 System register HAMAIRO bits [31:0] are architecturally mapped to AArch64 System register [AMAIR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HAMAIRO are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HAMAIRO is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

If an implementation does not provide any IMPLEMENTATION DEFINED memory attributes, this register is RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HAMAIRO

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HMAIR0;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HMAIR0;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1010 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HMAIR0 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HMAIR0 = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HMAIR1, Hyp Auxiliary Memory Attribute Indirection Register 1

The HMAIR1 characteristics are:

Purpose

Provides IMPLEMENTATION DEFINED memory attributes for the memory attribute encodings defined by [HMAIR1](#). These IMPLEMENTATION DEFINED attributes can only provide additional qualifiers for the memory attribute encodings, and cannot change the memory attributes defined in [HMAIR1](#).

Configuration

AArch32 System register HMAIR1 bits [31:0] are architecturally mapped to AArch64 System register [AMAIR_EL2\[63:32\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HMAIR1 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HMAIR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

If an implementation does not provide any IMPLEMENTATION DEFINED memory attributes, this register is RES0.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HMAIR1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1010 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HMAIR1;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HMAIR1;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1010 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HMAIR1 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HMAIR1 = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HCPTR, Hyp Architectural Feature Trap Register

The HCPTR characteristics are:

Purpose

Controls:

- Trapping to Hyp mode of Non-secure access, at EL1 or EL0, to trace, and to Advanced SIMD and floating-point functionality.
- Hyp mode access to trace, and to Advanced SIMD and floating-point functionality.

Note

Accesses to this functionality:

- From Non-secure modes other than Hyp mode are also affected by settings in the [CPACR](#) and [NSACR](#).
- From Hyp mode are also affected by settings in the [NSACR](#).

Exceptions generated by the [CPACR](#) and [NSACR](#) controls are higher priority than those generated by the HCPTR controls.

Configuration

AArch32 System register HCPTR bits [31:0] are architecturally mapped to AArch64 System register [CPTR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HCPTR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HCPTR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|------|----|----|----|----|----|-----|----|------|----|----|----|----------|----|------|----|-------|----|-------|----|------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TCPACTAM | | RES0 | | | | | | TTA | | RES0 | | | | TASERES0 | | RES1 | | TCP11 | | TCP10 | | RES1 | | | | | | | | | |

TCPAC, bit [31]

Traps Non-secure EL1 accesses to the [CPACR](#) to Hyp mode.

| TCPAC | Meaning |
|-------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 accesses to the CPACR are trapped to Hyp mode. |

Note

The [CPACR](#) is not accessible at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TAM, bit [30]

When FEAT_AMUv1 is implemented:

Trap Activity Monitor access. Traps Non-secure EL1 and EL0 accesses to all Activity Monitor registers to EL2.

| TAM | Meaning |
|-----|---|
| 0b0 | Accesses from Non-secure EL1 and EL0 to Activity Monitor registers are not trapped. |
| 0b1 | Accesses from Non-secure EL1 and EL0 to Activity Monitor registers are trapped to Hyp mode. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [29:21]

Reserved, RES0.

TTA, bit [20]

Traps Non-secure System register accesses to all implemented trace registers to Hyp mode.

| TTA | Meaning |
|-----|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any Non-secure System register access to an implemented trace register is trapped to Hyp mode, unless the access is trapped to EL1 by a CPACR or NSACR control, or the access is from Non-secure EL0 and the definition of the register in the appropriate trace architecture specification indicates that the register is not accessible from EL0. A trapped instruction generates: <ul style="list-style-type: none"> A Hyp Trap exception, if the exception is taken from Non-secure EL0 or EL1. An Undefined Instruction exception taken to Hyp mode, if the exception is taken from Hyp mode. |

If the implementation does not include a PE trace unit, or does not include a System register interface to the PE trace unit registers, it is IMPLEMENTATION DEFINED whether this bit:

- Is RES0.
- Is RES1.
- Can be written from Hyp mode, and from Secure Monitor mode when [SCR](#).NS is 1.

If EL3 is implemented and is using AArch32, and the value of [NSACR](#).NSTRCDIS is 1, in Non-secure state this field behaves as RAO/WI, regardless of its actual value.

Note

- The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the PE trace unit implements FEAT_ETMv4 or FEAT_ETE, EL0 accesses to the trace registers are UNDEFINED, and a resulting Undefined Instruction exception is higher priority than a HCPTR.TTA Hyp Trap exception.
- The Arm architecture does not provide traps on trace register accesses through the optional memory-mapped debug interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Bits [19:16]

Reserved, RES0.

TASE, bit [15]

Traps Non-secure execution of Advanced SIMD instructions to Hyp mode when the value of HCPTR.TCP10 is 0.

| TASE | Meaning |
|------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | When the value of HCPTR.TCP10 is 0, any attempt to execute an Advanced SIMD instruction in Non-secure state is trapped to Hyp mode, unless it is trapped to EL1 by a CPACR or NSACR control. A trapped instruction generates: <ul style="list-style-type: none"> A Hyp Trap exception, if the exception is taken from Non-secure EL0 or EL1. An Undefined Instruction exception taken to Hyp mode, if the exception is taken from Hyp mode. |

When the value of HCPTR.TCP10 is 1, the value of this field is ignored.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES1. Otherwise, it is IMPLEMENTATION DEFINED whether this field is implemented as a RW field. If it is not implemented as a RW field, then it is RAZ/WI.

If EL3 is implemented and is using AArch32, and the value of [NSACR](#).NSASEDIS is 1, in Non-secure state this field behaves as RAO/WI, regardless of its actual value. This applies even if the field is implemented as RAZ/WI.

For the list of instructions affected by this field, see 'Controls of Advanced SIMD operation that do not apply to floating-point operation'.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Bit [14]

Reserved, RES0.

Bits [13:12]

Reserved, RES1.

TCP11, bit [11]

The value of this field is ignored. If this field is programmed with a different value to the TCP10 bit then this field is UNKNOWN on a direct read of the HCPTR.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES1.

If EL3 is implemented and is using AArch32, and the value of [NSACR](#).cp10 is 0, in Non-secure state this field behaves as RAO/WI, regardless of its actual value.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TCP10, bit [10]

Trap Non-secure accesses to Advanced SIMD and floating-point functionality to Hyp mode:

| TCP10 | Meaning |
|-------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any attempted access to Advanced SIMD and floating-point functionality from Non-secure state is trapped to Hyp mode, unless it is trapped to EL1 by a CPACR or NSACR control. A trapped instruction generates: <ul style="list-style-type: none"> • A Hyp Trap exception, if the exception is taken from Non-secure EL0 or EL1. • An Undefined Instruction exception taken to Hyp mode, if the exception is taken from Hyp mode. |

The Advanced SIMD and floating-point features controlled by these fields are:

- Execution of any floating-point or Advanced SIMD instruction.
- Any access to the Advanced SIMD and floating-point registers D0-D31 and their views as S0-S31 and Q0-Q15.
- Any access to the [FPSCR](#), [FPSID](#), [MVFR0](#), [MVFR1](#), [MVFR2](#), or [FPEXC](#) System registers.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES1.

If EL3 is implemented and is using AArch32, and the value of [NSACR](#).cp10 is 0, in Non-secure state this field behaves as RAO/WI, regardless of its actual value.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Bits [9:0]

Reserved, RES1.

Accessing HCPTR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return HCPTR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HCPTR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TCPAC == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        HCPTR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HCPTR = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HCR, Hyp Configuration Register

The HCR characteristics are:

Purpose

Provides configuration controls for virtualization, including defining whether various Non-secure operations are trapped to Hyp mode.

Configuration

AArch32 System register HCR bits [31:0] are architecturally mapped to AArch64 System register [HCR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|-----|------|-----|-----|------|-----|-----|-----|------|-------|------|------|------|------|------|------|------|-------|----|------|------|-----|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 |
| RES0 | TRVM | HCD | RES0 | TGE | TVM | TTLB | TPU | TPC | TSW | TACT | TIDCP | TSCT | TID3 | TID2 | TID1 | TID0 | TWET | TWID | DCBSU | FB | VAVI | VFAM | OMC | | | | |

Bit [31]

Reserved, RES0.

TRVM, bit [30]

Trap Reads of Virtual Memory controls. Traps Non-secure EL1 reads of the virtual memory control registers to EL2, when EL2 is enabled in the current Security state.

The registers for which read accesses are trapped are as follows:

[SCTLR](#), [TTBR0](#), [TTBR1](#), [TTBCR](#), [TTBCR2](#), [DACR](#), [DFSR](#), [IFSR](#), [DFAR](#), [IFAR](#), [ADFSR](#), [AIFSR](#), [PRRR](#), [NMRR](#), [MAIR0](#), [MAIR1](#), [AMAIRO](#), [AMAIR1](#), [CONTEXTIDR](#).

| TRVM | Meaning |
|------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 read accesses to the specified Virtual Memory controls are trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

HCD, bit [29]

When EL3 is not implemented:

HVC instruction disable. Disables Non-secure EL1 and EL2 execution of HVC instructions, when EL2 is enabled in the current Security state.

| HCD | Meaning |
|-----|--|
| 0b0 | HVC instruction execution is enabled at EL2 and EL1. |
| 0b1 | HVC instructions are UNDEFINED at EL2 and Non-secure EL1. The Undefined Instruction exception is taken to the Exception level at which the HVC instruction is executed. |

Note

HVC instructions are always UNDEFINED at EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [28]

Reserved, RES0.

TGE, bit [27]

Trap General Exceptions, from Non-secure EL0.

| TGE | Meaning |
|-----|---|
| 0b0 | This control has no effect on execution at EL0. |
| 0b1 | When EL2 is not enabled in the current Security state, this control has no effect on execution at EL0. When EL2 is enabled in the current Security state, then: <ul style="list-style-type: none"> All exceptions that would be routed to EL1 are routed to EL2. The SCTLR.M bit is treated as being 0 for all purposes other than returning the result of a direct read of SCTLR. The HCR.{FMO, IMO, AMO} bits are treated as being 1 for all purposes other than returning the result of a direct read of HCR. All virtual interrupts are disabled. Any IMPLEMENTATION DEFINED mechanisms for signaling virtual interrupts are disabled. An exception return to EL1 is treated as an illegal exception return. Monitor mode execution of an MSR or CPS instruction that changes PSTATE.M to a Non-secure EL1 mode is an illegal change to PSTATE.M. For more information see 'Illegal changes to PSTATE.M'. |

Also, when HCR.TGE is 1:

- If EL3 is using AArch32, an attempt to change from a Secure PL1 mode to a Non-secure EL1 mode by changing [SCR.NS](#) from 0 to 1 results in [SCR.NS](#) remaining as 0.
- The [HDCR](#).{TDRA, TDOSA, TDA, TDE} bits are ignored and treated as being 1 other than for the purpose of a direct read of [HDCR](#).

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TVM, bit [26]

Trap Virtual Memory controls. Traps Non-secure EL1 writes to the virtual memory control registers to EL2, when EL2 is enabled in the current Security state.

The registers for which write accesses are trapped are as follows:

[SCTLR](#), [TTBR0](#), [TTBR1](#), [TTBCR](#), [TTBCR2](#), [DACR](#), [DFSR](#), [IFSR](#), [DFAR](#), [IFAR](#), [ADFSR](#), [AIFSR](#), [PRRR](#), [NMRR](#), [MAIRO](#), [MAIR1](#), [AMAIRO](#), [AMAIR1](#), [CONTEXTIDR](#).

| TVM | Meaning |
|------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 write accesses to the specified virtual memory control registers are trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TTLB, bit [25]

Trap TLB maintenance instructions. Traps Non-secure EL1 execution of a TLBI instruction to EL2, when EL2 is enabled in the current Security state.

This applies to the following instructions:

[TLBIALLIS](#), [TLBIMVAIS](#), [TLBIASIDIS](#), [TLBIMVAAIS](#), [TLBIMVALIS](#), [TLBIMVAALIS](#), [ITLBIALL](#), [ITLBIMVA](#), [ITLBIASID](#), [DTLBIALL](#), [DTLBIMVA](#), [DTLBIASID](#), [TLBIALL](#), [TLBIMVA](#), [TLBIASID](#), [TLBIMVAA](#), [TLBIMVAL](#), [TLBIMVAAL](#)

| TTLB | Meaning |
|-------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 accesses to the specified TLB maintenance instructions are trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TPU, bit [24]

Trap cache maintenance instructions that operate to the Point of Unification. Traps Non-secure EL1 execution of those cache maintenance instructions to EL2, when EL2 is enabled in the current Security state.

This applies to the following instructions:

- [ICIMVAU](#), [ICIALLU](#), [ICIALLUIS](#), [DCCMVAU](#).

Note

An Undefined Instruction exception generated at EL0 is higher priority than this trap to EL2, and these instructions are always UNDEFINED at EL0.

| TPU | Meaning |
|------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 execution of the specified cache maintenance instructions is trapped to EL2. |

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TPC, bit [23]

Trap data or unified cache maintenance instructions that operate to the Point of Coherency. Traps Non-secure EL1 execution of those cache maintenance instructions to EL2, when EL2 is enabled in the current Security state.

This applies to the following instructions:

- [DCIMVAC](#), [DCCIMVAC](#), [DCCMVAC](#).

Note

An Undefined Instruction exception generated at EL0 is higher priority than this trap to EL2, and these instructions are always UNDEFINED at EL0.

| TPC | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 execution of the specified cache maintenance instructions is trapped to EL2. |

If the Point of Coherency is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean, invalidate, or clean and invalidate instruction that operates by VA to the point of coherency can be trapped when the value of this control is 1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TSW, bit [22]

Trap data or unified cache maintenance instructions that operate by Set/Way. Traps Non-secure EL1 execution of those cache maintenance instructions by set/way to EL2, when EL2 is enabled in the current Security state.

This applies to the following instructions:

- [DCISW](#), [DCCSW](#), [DCCISW](#).

Note

An Undefined Instruction exception generated at EL0 is higher priority than this trap to EL2, and these instructions are always UNDEFINED at EL0.

| TSW | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 execution of the specified cache maintenance instructions is trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TAC, bit [21]

Trap Auxiliary Control Registers. Traps Non-secure EL1 accesses to the Auxiliary Control Registers to EL2, when EL2 is enabled in the current Security state, from both Execution states.

This applies to the following register accesses:

[ACTLR](#) and, if implemented, [ACTLR2](#).

| TAC | Meaning |
|-----|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 accesses to the specified registers are trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TIDCP, bit [20]

Trap IMPLEMENTATION DEFINED functionality. Traps Non-secure EL1 accesses to the encodings for IMPLEMENTATION DEFINED System Registers to EL2, when EL2 is enabled in the current Security state.

MCR and MRC instructions accessing the following encodings:

- All coproc==p15, CRn==c9, Opcode1 = {0-7}, CRm == {c0-c2, c5-c8}, opcode2 == {0-7}.
- All coproc==p15, CRn==c10, Opcode1 =={0-7}, CRm == {c0, c1, c4, c8}, opcode2 == {0-7}.
- All coproc==p15, CRn==c11, Opcode1=={0-7}, CRm == {c0-c8, c15}, opcode2 == {0-7}.

When HCR.TIDCP is set to 1, it is IMPLEMENTATION DEFINED whether any of this functionality accessed from Non-secure EL0 is trapped to EL2. Otherwise, it is UNDEFINED and the PE takes an Undefined Instruction exception to Non-secure Undefined mode.

| TIDCP | Meaning |
|-------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 accesses to the specified System register encodings for IMPLEMENTATION DEFINED functionality are trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TSC, bit [19]

Trap SMC instructions. Traps Non-secure EL1 execution of SMC instructions to Hyp mode.

| TSC | Meaning |
|-----|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any attempt to execute an SMC instruction at Non-secure EL1 is trapped to Hyp mode, regardless of the value of SCR.SCD . |

The Armv8-A architecture permits, but does not require, this trap to apply to conditional SMC instructions that fail their condition code check, in the same way as with traps on other conditional instructions.

Note

- This trap is only implemented if the implementation includes EL3.
- SMC instructions are always UNDEFINED at PL0.
- This bit traps execution of the SMC instruction. It is not a routing control for the SMC exception. Hyp Trap exceptions and SMC exceptions have different preferred return addresses.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TID3, bit [18]

Trap ID group 3. Traps Non-secure EL1 reads of the following registers to EL2, when EL2 is enabled in the current Security state as follows:

- VMRS access to [MVFR0](#), [MVFR1](#), and [MVFR2](#), reported using EC syndrome value 0x08, unless access is also trapped by [HCPTR](#) which takes priority.
- MRC access to the following registers are reported using EC syndrome value 0x03:
 - [ID_PFR0](#), [ID_PFR1](#), [ID_PFR2](#), [ID_DFR0](#), [ID_AFR0](#), [ID_MMFR0](#), [ID_MMFR1](#), [ID_MMFR2](#), [ID_MMFR3](#), [ID_ISAR0](#), [ID_ISAR1](#), [ID_ISAR2](#), [ID_ISAR3](#), [ID_ISAR4](#), and [ID_ISAR5](#).
 - If FEAT_FGT is implemented:
 - [ID_MMFR4](#) and [ID_MMFR5](#) are trapped to EL2.
 - [ID_ISAR6](#) is trapped to EL2.
 - [ID_DFR1](#) is trapped to EL2.

- This field traps all MRC accesses to registers in the following range that are not already mentioned in this field description: coproc == p15, opc1 == 0, CRn == c0, CRm == {c2-c7}, opc2 == {0-7}.
- If FEAT_FGT is not implemented:
 - [ID_MMFR4](#) and [ID_MMFR5](#) are trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to [ID_MMFR4](#) or [ID_MMFR5](#) are trapped.
 - [ID_ISAR6](#) is trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to [ID_ISAR6](#) are trapped to EL2.
 - [ID_DFR1](#) is trapped to EL2, unless implemented as RAZ, when it is IMPLEMENTATION DEFINED whether accesses to [ID_DFR1](#) are trapped to EL2.
 - Otherwise, it is IMPLEMENTATION DEFINED whether this bit traps MRC accesses to registers not already mentioned, with coproc == p15, opc1 == 0, CRn == c0, CRm == {c2-c7}, opc2 == {0-7}.

| TID3 | Meaning |
|------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | The specified Non-secure EL1 read accesses to ID group 3 registers are trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TID2, bit [17]

Trap ID group 2. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:

- Non-secure EL1 and EL0 reads of the [CTR](#), [CCSIDR](#), [CCSIDR2](#), [CLIDR](#), and [CSSELR](#).
- Non-secure EL1 and EL0 writes to the [CSSELR](#).

| TID2 | Meaning |
|------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | The specified Non-secure EL1 and EL0 accesses to ID group 2 registers are trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TID1, bit [16]

Trap ID group 1. Traps Non-secure EL1 reads of the following registers to EL2, when EL2 is enabled in the current Security state:

[TCMTR](#), [TLBTR](#), [REVIDR](#), [AIDR](#).

| TID1 | Meaning |
|------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | The specified Non-secure EL1 read accesses to ID group 1 registers are trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TID0, bit [15]

Trap ID group 0. Traps the following register accesses to EL2, when EL2 is enabled in the current Security state:

- Non-secure EL1 reads of the [JIDR](#) and [FPSID](#).
- If the [JIDR](#) is RAZ from Non-secure EL0, Non-secure EL0 reads of the [JIDR](#).

Note

- It is IMPLEMENTATION DEFINED whether the [JIDR](#) is RAZ or UNDEFINED at EL0. If it is UNDEFINED at EL0 then the Undefined Instruction exception takes precedence over this trap.
- The [FPSID](#) is not accessible at EL0.
- Writes to the [FPSID](#) are ignored, and not trapped by this control.

| TID0 | Meaning |
|------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | The specified Non-secure EL1 read accesses to ID group 0 registers are trapped to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TWE, bit [14]

Traps Non-secure EL0 and EL1 execution of WFE instructions to EL2, when EL2 is enabled in the current Security state.

| TWE | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any attempt to execute a WFE instruction at Non-secure EL0 or EL1 is trapped to EL2, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWE . |

The attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE can complete at any time, even without a Wakeup event, the traps on WFE are not guaranteed to be taken, even if the WFE is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TWI, bit [13]

Traps Non-secure EL0 and EL1 execution of WFI instructions to EL2, when EL2 is enabled in the current Security state.

| TWI | Meaning |
|-----|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any attempt to execute a WFI instruction at Non-secure EL0 or EL1 is trapped to EL2, if the instruction would otherwise have caused the PE to enter a low-power state and it is not trapped by SCTLR.nTWI . |

The attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFI can complete at any time, even without a Wakeup event, the traps on WFI are not guaranteed to be taken, even if the WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

DC, bit [12]

Default Cacheability.

| DC | Meaning |
|-----|--|
| 0b0 | This control has no effect on the Non-secure EL1&0 translation regime. |
| 0b1 | In Non-secure state: <ul style="list-style-type: none"> The SCTLR.M field behaves as 0 for all purposes other than a direct read of the value of the field. The HCR.VM field behaves as 1 for all purposes other than a direct read of the value of the field. The memory type produced by the first stage of the EL1&0 translation regime is Normal Non-Shareable, Inner Write-Back Read-Allocate Write-Allocate, Outer Write-Back Read-Allocate Write-Allocate. |

This field has no effect on the EL2 and EL3 translation regimes.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

BSU, bits [11:10]

Barrier Shareability upgrade. This field determines the minimum shareability domain that is applied to any barrier instruction executed from Non-secure EL1 or Non-secure EL0:

| BSU | Meaning |
|------|------------------|
| 0b00 | No effect. |
| 0b01 | Inner Shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Full system. |

This value is combined with the specified level of the barrier held in its instruction, using the same principles as combining the shareability attributes from two stages of address translation.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

FB, bit [9]

Force broadcast. Causes the following instructions to be broadcast within the Inner Shareable domain when executed from Non-secure EL1:

[BPIALL](#), [TLBIALL](#), [TLBIMVA](#), [TLBIASID](#), [DTLBIALL](#), [DTLBIMVA](#), [DTLBIASID](#), [ITLBIALL](#), [ITLBIMVA](#), [ITLBIASID](#), [TLBIMVAA](#), [ICIALLU](#), [TLBIMVAL](#), [TLBIMVAAL](#).

| FB | Meaning |
|-----|---|
| 0b0 | This field has no effect on the operation of the specified instructions. |
| 0b1 | When one of the specified instruction is executed at Non-secure EL1, the instruction is broadcast within the Inner Shareable shareability domain. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

VA, bit [8]

Virtual SError interrupt exception.

| VA | Meaning |
|-----|--|
| 0b0 | This mechanism is not making a virtual SError interrupt pending. |
| 0b1 | A virtual SError interrupt is pending because of this mechanism. |

The virtual SError interrupt is enabled only when the value of HCR.{TGE, AMO} is {0, 1}.

The Guest OS cannot distinguish the virtual exception from the corresponding physical exception.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

VI, bit [7]

Virtual IRQ exception.

| VI | Meaning |
|-----|---|
| 0b0 | This mechanism is not making a virtual IRQ pending. |
| 0b1 | A virtual IRQ is pending because of this mechanism. |

The virtual IRQ is enabled only when the value of HCR.{TGE, IMO} is {0, 1}.

The Guest OS cannot distinguish the virtual exception from the corresponding physical exception.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

VF, bit [6]

Virtual FIQ exception.

| VF | Meaning |
|-----|---|
| 0b0 | This mechanism is not making a virtual FIQ pending. |
| 0b1 | A virtual FIQ is pending because of this mechanism. |

The virtual FIQ is enabled only when the value of HCR.{TGE, FMO} is {0, 1}.

The Guest OS cannot distinguish the virtual exception from the corresponding physical exception.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

AMO, bit [5]

SError interrupt Mask Override. When this bit is set to 1, it overrides the effect of PSTATE.A, and enables virtual exception signaling by the VA bit.

If the value of HCR.TGE is 0, then virtual SError interrupts are enabled in Non-secure state.

If the value of HCR.TGE is 1, then in Non-secure state the HCR.AMO bit behaves as 1 for all purposes other than a direct read of the value of the bit.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

IMO, bit [4]

IRQ Mask Override. When this bit is set to 1, it overrides the effect of PSTATE.I, and enables virtual exception signaling by the VI bit.

If the value of HCR.TGE is 0, then Virtual IRQ interrupts are enabled in the Non-secure state.

If the value of HCR.TGE is 1, then in Non-secure state the HCR.IMO bit behaves as 1 for all purposes other than a direct read of the value of the bit.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

FMO, bit [3]

FIQ Mask Override. When this bit is set to 1, it overrides the effect of PSTATE.F, and enables virtual exception signaling by the VF bit.

If the value of HCR.TGE is 0, then Virtual FIQ interrupts are enabled in the Non-secure state.

If the value of HCR.TGE is 1, then in Non-secure state the HCR.FMO bit behaves as 1 for all purposes other than a direct read of the value of the bit.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

PTW, bit [2]

Protected Table Walk. In the Non-secure PL1&0 translation regime, a translation table access made as part of a stage 1 translation table walk is subject to a stage 2 translation. The combining of the memory type attributes from the two stages of translation means the access might be made to a type of Device memory. If this occurs then the value of this bit determines the behavior:

| PTW | Meaning |
|-----|--|
| 0b0 | The translation table walk occurs as if it is to Normal Non-cacheable memory. This means it can be made speculatively. |
| 0b1 | The memory access generates a stage 2 Permission fault. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

SWIO, bit [1]

Set/Way Invalidation Override. Causes Non-secure EL1 execution of the data cache invalidate by set/way instructions to perform a data cache clean and invalidate by set/way.

| SWIO | Meaning |
|------|---|
| 0b0 | This control has no effect on the operation of data cache invalidate by set/way instructions. |
| 0b1 | Data cache invalidate by set/way instructions perform a data cache clean and invalidate by set/way. |

When this bit is set to 1, [DCISW](#) performs the same invalidation as a [DCCISW](#) instruction.

As a result of changes to the behavior of [DCISW](#), this bit is redundant in Armv8. This bit can be implemented as RES1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

VM, bit [0]

Virtualization enable. Enables stage 2 address translation for the Non-secure EL1&0 translation regime.

| VM | Meaning |
|-----|--|
| 0b0 | Non-secure EL1&0 stage 2 address translation disabled. |
| 0b1 | Non-secure EL1&0 stage 2 address translation enabled. |

If the HCR.DC bit is set to 1, then the behavior of the PE when executing in a Non-secure mode other than Hyp mode is consistent with HCR.VM being 1, regardless of the actual value of HCR.VM, other than the value returned by an explicit read of HCR.VM.

When the value of this bit is 1, data cache invalidate instructions executed at Non-secure EL1 perform a data cache clean and invalidate. For the invalidate by set/way instruction this behavior applies regardless of the value of the HCR.SWIO bit.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Accessing HCR

Accesses to this register use the following encodings in the System register encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HCR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HCR = R[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HCR2, Hyp Configuration Register 2

The HCR2 characteristics are:

Purpose

Provides additional configuration controls for virtualization.

Configuration

AArch32 System register HCR2 bits [31:0] are architecturally mapped to AArch64 System register [HCR_EL2\[63:32\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HCR2 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HCR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|------|--------|------|-----|------|-------|------|----|----|----|----|----|----|---|---|---|------|-----|-----|-----|------|------|-----|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | | | | | | | | | RES0 | TTLBIS | RES0 | TOC | RES0 | TICAB | TID4 | | | | | | | | | | RES0 | MIO | CNC | TEA | TERR | RES0 | IDC | D |

Bits [31:23]

Reserved, RES0.

TTLBIS, bit [22]

When FEAT_EVT is implemented:

Trap TLB maintenance instructions that operate on the Inner Shareable domain. Traps execution of the following TLB maintenance instructions at EL1 to EL2:

[TLBIALIS](#), [TLBIMVAIS](#), [TLBIASIDIS](#), [TLBIMVAAIS](#), [TLBIMVALIS](#), [TLBIMVAALIS](#)

| TTLBIS | Meaning |
|--------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 execution of the specified TLB maintenance instructions is trapped to EL2. |

When FEAT_VHE and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

Otherwise:

Reserved, RES0.

Bit [21]

Reserved, RES0.

TOCU, bit [20]**When FEAT_EVT is implemented:**

Trap cache maintenance instructions that operate to the Point of Unification. Traps execution of those cache maintenance instructions at EL1 or EL0 using AArch64, and at EL1 using AArch32, to EL2.

This applies to the following instructions:

- When Non-secure EL0 is using AArch64, [IC IVAU](#), [DC CVAU](#). However, if the value of [SCTLR_EL1](#).UCI is 0 these instructions are UNDEFINED at EL0 and any resulting exception is higher priority than this trap to EL2.
- When EL1 is using AArch64, [IC IVAU](#), [IC IALLU](#), [DC CVAU](#).
- When Non-secure EL1 is using AArch32, [ICIMVAU](#), [ICIALLU](#), [DCCMVAU](#).

Note

An exception generated because an instruction is UNDEFINED at EL0 is higher priority than this trap to EL2. In addition:

- [IC IALLUIS](#) and [IC IALLU](#) are always UNDEFINED at EL0 using AArch64.
- [ICIMVAU](#), [ICIALLU](#), [ICIALLUIS](#), and [DCCMVAU](#) are always UNDEFINED at EL0 using AArch32.

| TOCU | Meaning |
|------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure execution of the specified cache maintenance instructions is trapped to EL2. |

If the Point of Unification is before any level of data cache, it is IMPLEMENTATION DEFINED whether the execution of any data or unified cache clean by VA to the Point of Unification instruction can be trapped when the value of this control is 1.

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When FEAT_VHE is implemented, and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

Otherwise:

Reserved, RES0.

Bit [19]

Reserved, RES0.

TICAB, bit [18]**When FEAT_EVT is implemented:**

Trap ICIALLUIS cache maintenance instructions. Traps execution of those cache maintenance instructions at EL1 to EL2.

This applies to the following instructions:

[ICIALLUIS](#).

| TICAB | Meaning |
|-------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 execution of the specified cache maintenance instructions is trapped to EL2. |

If the Point of Unification is before any level of instruction cache, it is IMPLEMENTATION DEFINED whether the execution of any instruction cache invalidate to the Point of Unification instruction can be trapped when the value of this control is 1.

When FEAT_VHE and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

Otherwise:

Reserved, RES0.

TID4, bit [17]

When FEAT_EVT is implemented:

Trap ID group 4. Traps the following register accesses to EL2:

- EL1 reads of [CCSIDR](#), [CCSIDR2](#), [CLIDR](#), and [CSSELR](#).
- EL1 writes to [CSSELR](#).

| TID4 | Meaning |
|------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | The specified Non-secure EL1 and EL0 accesses to ID group 4 registers are trapped to EL2. |

When FEAT_VHE is implemented and the value of [HCR_EL2](#).{E2H, TGE} is {1, 1}, this field behaves as 0 for all purposes other than a direct read of the value of this bit.

Otherwise:

Reserved, RES0.

Bits [16:7]

Reserved, RES0.

MIOCNCNCE, bit [6]

Mismatched Inner/Outer Cacheable Non-Coherency Enable, for the Non-secure PL1&0 translation regime.

| MIOCNCNCE | Meaning |
|-----------|--|
| 0b0 | For the Non-secure PL1&0 translation regime, for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there must be no loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute. |
| 0b1 | For the Non-secure PL1&0 translation regime, for permitted accesses to a memory location that use a common definition of the Shareability and Cacheability of the location, there might be a loss of coherency if the Inner Cacheability attribute for those accesses differs from the Outer Cacheability attribute. |

For more information, see 'Mismatched memory attributes'.

This field can be implemented as RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to an architecturally UNKNOWN value.

TEA, bit [5]

When FEAT_RAS is implemented:

Route synchronous External abort exceptions from EL0 and EL1 to EL2.

| TEA | Meaning |
|-----|---|
| 0b0 | Does not route synchronous External abort exceptions from Non-secure EL0 and EL1 to EL2. |
| 0b1 | Route synchronous External abort exceptions from Non-secure EL0 and EL1 to EL2, if not routed to EL3. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

TERR, bit [4]

When FEAT_RAS is implemented:

Trap Error record accesses from EL1 to EL2. Trap accesses to the following registers from EL1 to EL2:

[ERRIDR](#), [ERRSELR](#), [ERXADDR](#), [ERXADDR2](#), [ERXCTLR](#), [ERXCTLR2](#), [ERXFR](#), [ERXFR2](#), [ERXMISC0](#), [ERXMISC1](#), [ERXMISC2](#), [ERXMISC3](#), and [ERXSTATUS](#).

When FEAT_RASv1p1 is implemented, [ERXMISC4](#), [ERXMISC5](#), [ERXMISC6](#), and [ERXMISC7](#).

| TERR | Meaning |
|------|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Accesses to the specified registers from EL1 generate a Trap exception to EL2. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [3:2]

Reserved, RES0.

ID, bit [1]

Stage 2 Instruction access cacheability disable. For the Non-secure PL1&0 translation regime, when [HCR.VM](#)==1, this control forces all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable.

| ID | Meaning |
|-----|---|
| 0b0 | This control has no effect on stage 2 of the Non-secure PL1&0 translation regime. |
| 0b1 | For the Non-secure PL1&0 translation regime, forces all stage 2 translations for instruction accesses to Normal memory to be Non-cacheable. |

This bit has no effect on the EL2 translation regime.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

CD, bit [0]

Stage 2 Data access cacheability disable. When [HCR.VM](#)==1, this forces all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable for the Non-secure PL1&0 translation regime.

| CD | Meaning |
|-----|--|
| 0b0 | This control has no effect on stage 2 of the Non-secure PL1&0 translation regime for data accesses and translation table walks. |
| 0b1 | For the Non-secure PL1&0 translation regime, forces all stage 2 translations for data accesses and translation table walks to Normal memory to be Non-cacheable. |

This bit has no effect on the EL2 translation regime.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Accessing HCR2

Accesses to this register use the following encodings in the System register encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HCR2;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HCR2;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b100 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HCR2 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HCR2 = R[t];
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HDCR, Hyp Debug Control Register

The HDCR characteristics are:

Purpose

Controls the trapping to Hyp mode of Non-secure accesses, at EL1 or lower, to functions provided by the debug and trace architectures and the Performance Monitors Extension.

Configuration

AArch32 System register HDCR bits [31:0] are architecturally mapped to AArch64 System register [MDCR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HDCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3, and other than for a direct read of the register, the PE behaves as if `HDCR.HPMN == PMCR.N`.

Attributes

HDCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|--------|-------|------|-----|------|------|------|------|------|------|------|------|------|------|------|-------|-----|-------|----|----|----|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 |
| RES0 | HPMFZO | MTPME | TDCC | HLP | RES0 | HCCD | RES0 | TTRF | RES0 | HPMD | RES0 | TDRA | DOSA | TDAT | DEHP | MPMET | TPM | TPMCR | | | | | | | | | |

Bits [31:30]

Reserved, RES0.

HPMFZO, bit [29]

When `FEAT_PMUv3p7` is implemented:

Hyp Performance Monitors Freeze-on-overflow. Stop event counters on overflow.

| HPMFZO | Meaning |
|--------|---|
| 0b0 | Do not freeze on overflow. |
| 0b1 | Event counters do not count when PMOVSr[(PMCR.N-1):HDCR.HPMN] is nonzero. |

If `HDCR.HPMN` is less than [PMCR.N](#), this field affects the operation of event counters in the range [`HDCR.HPMN` .. ([PMCR.N](#)-1)].

This field does not affect the operation of other event counters and [PMCCNTR](#).

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MTPME, bit [28]**When FEAT_MTPMU is implemented and EL3 is not implemented:**

Multi-threaded PMU Enable. Enables use of the [PMEVTYPER<n>](#).MT bits.

| MTPME | Meaning |
|-------|--|
| 0b0 | FEAT_MTPMU is disabled. The Effective value of PMEVTYPER<n> .MT is zero. |
| 0b1 | PMEVTYPER<n> .MT bits not affected by this bit. |

If FEAT_MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is IMPLEMENTATION DEFINED whether the PE behaves as if this bit is 0b0.

The reset behavior of this field is:

- On a Cold reset, in a system where the PE resets into EL2 or EL3, this field resets to 1.

Otherwise:

Reserved, RES0.

TDCC, bit [27]**When FEAT_FGT is implemented:**

Trap DCC. Traps use of the Debug Comms Channel at EL1 and EL0 to EL2.

| TDCC | Meaning |
|------|---|
| 0b0 | This control does not cause any register accesses to be trapped. |
| 0b1 | If EL2 is implemented and enabled in the current Security state, accesses to the DCC registers at EL1 and EL0 generate a Hyp Trap exception, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state. |

The DCC registers trapped by this control are:

- [DBGDTRRXext](#), [DBGDTRTXext](#), [DBGDSCRint](#), [DBGDCCINT](#), and, when the PE is in Non-debug state, [DBGDTRRXint](#) and [DBGDTRTXint](#).

The traps are reported with EC syndrome value:

- 0x05 for trapped MRC and MCR accesses with coproc == 0b1110.
- 0x06 for trapped LDC to [DBGDTRTXint](#) and STC from [DBGDTRRXint](#).

When the PE is in Debug state, HDCR.TDCC does not trap any accesses to:

- [DBGDTRRXint](#) and [DBGDTRTXint](#).

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

HLP, bit [26]**When FEAT_PMUv3p5 is implemented:**

Hypervisor Long event counter enable. Determines when unsigned overflow is recorded by an event counter overflow bit.

| HLP | Meaning |
|-----|--|
| 0b0 | Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n> [31:0]. |
| 0b1 | Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n> [63:0]. |

If the highest implemented Exception level is using AArch32, it is IMPLEMENTATION DEFINED whether this bit is read/write or RAZ/WI.

If HDCR.HPMN is less than PMCR.N, this bit affects the operation of event counters in the range [HDCR.HPMN..[\(PMCR.N-1\)](#)].

This field does not affect the operation of other event counters.

The operation of this field applies even when EL2 is disabled in the current Security state.

Note

[PMEVCNTR<n>](#)[63:32] cannot be accessed directly in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [25:24]

Reserved, RES0.

HCCD, bit [23]

When FEAT_PMUv3p5 is implemented:

Hypervisor Cycle Counter Disable. Prohibits [PMCCNTR](#) from counting at EL2.

| HCCD | Meaning |
|------|--|
| 0b0 | Cycle counting by PMCCNTR is not affected by this mechanism. |
| 0b1 | Cycle counting by PMCCNTR is prohibited at EL2. |

This field does not affect the CPU_CYCLES event or any other event that counts cycles.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [22:20]

Reserved, RES0.

TTRF, bit [19]

When FEAT_TRF is implemented:

Traps use of the Trace Filter Control registers at EL1 to EL2.

| TTRF | Meaning |
|------|--|
| 0b0 | Accesses to TRFCR at EL1 are not affected by this control bit. |
| 0b1 | Accesses to TRFCR at EL1 generate a Hyp Trap exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [18]

Reserved, RES0.

HPMD, bit [17]

When **FEAT_PMUv3p1** is implemented and **FEAT_Debugv8p2** is implemented:

Guest Performance Monitors Disable. Controls event counting by some event counters at EL2.

| HPMD | Meaning |
|------|--|
| 0b0 | Event counting and PMCCNTR are not affected by this mechanism. |
| 0b1 | Event counting by some event counters is prohibited in Hyp mode. If PMCR.DP is 1, PMCCNTR is disabled in Hyp mode. Otherwise, PMCCNTR is not affected by this mechanism. |

If HDCR.HPMN is not 0, this field affects the operation of event counters in the range [0 .. (HDCR.HPMN-1)].

This field does not affect the operation of other event counters.

If [PMCR.DP](#) is 1, this field affects [PMCCNTR](#).

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

When **FEAT_PMUv3p1** is implemented:

Guest Performance Monitors Disable. Controls event counting by some event counters at EL2.

| HPMD | Meaning |
|------|--|
| 0b0 | Event counting and PMCCNTR are not affected by this mechanism. |
| 0b1 | If <code>ExternalSecureNoninvasiveDebugEnabled()</code> is FALSE, event counting by some event counters is prohibited in Hyp mode, and if PMCR.DP is 1, PMCCNTR is disabled in Hyp mode. |

If `ExternalSecureNoninvasiveDebugEnabled()` is TRUE, this field does not affect the event counters and does not affect [PMCCNTR](#).

Otherwise:

- If HDCR.HPMN is not 0, this field affects the operation of event counters in the range [0 .. (HDCR.HPMN-1)].
- This field does not affect the operation of other event counters.
- If [PMCR.DP](#) is 1, this field affects [PMCCNTR](#).

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [16:12]

Reserved, RES0.

TDRA, bit [11]

Trap Debug ROM Address register access. Traps Non-secure EL0 and EL1 System register accesses to the Debug ROM registers to Hyp mode.

| TDRA | Meaning |
|------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL0 and EL1 System register accesses to the DBGDRAR or DBGDSAR are trapped to Hyp mode, unless it is trapped by DBGDSCRext.UDCCdis . |

If [HCR.TGE](#) or [HDCR.TDE](#) is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TDOSA, bit [10]

When **FEAT_DoubleLock** is implemented:

Trap debug OS-related register access. Traps Non-secure EL1 System register accesses to the powerdown debug registers to Hyp mode.

| TDOSA | Meaning |
|-------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 System register accesses to the powerdown debug registers are trapped to Hyp mode. |

The registers for which accesses are trapped are as follows:

- [DBGOSLSR](#), [DBGOSLAR](#), [DBGOSDLR](#), and [DBGPRCR](#).
- Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

Note

These registers are not accessible at EL0.

If [HCR.TGE](#) or [HDCR.TDE](#) is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Trap debug OS-related register access. Traps Non-secure EL1 System register accesses to the powerdown debug registers to Hyp mode.

| TDOSA | Meaning |
|-------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL1 System register accesses to the powerdown debug registers are trapped to Hyp mode. |

The registers for which accesses are trapped are as follows:

- [DBGOSLSR](#), [DBGOSLAR](#), and [DBGPRCR](#).
- Any IMPLEMENTATION DEFINED register with similar functionality that the implementation specifies as trapped by this bit.

It is IMPLEMENTATION DEFINED whether accesses to [DBGOSDLR](#) are trapped.

Note

These registers are not accessible at EL0.

If [HCR](#).TGE or [HDCR](#).TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TDA, bit [9]

Trap debug access. Traps Non-secure EL0 and EL1 System register accesses to those debug System registers in the (coproc==0b1110) encoding space that are not trapped by either of the following:

- [HDCR](#).TDRA.
- [HDCR](#).TDOSA.

| TDA | Meaning |
|-----|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL0 or EL1 System register accesses to the debug registers, other than the registers trapped by HDCR .TDRA and HDCR .TDOSA, are trapped to Hyp mode, unless it is trapped by DBGDSCRext .UDCCdis. |

Traps of AArch32 accesses to [DBGDTRRXint](#) and [DBGDTRTXint](#) are ignored in Debug state.

If [HCR](#).TGE or [HDCR](#).TDE is 1, behavior is as if this bit is 1 other than for the purpose of a direct read.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

TDE, bit [8]

Trap Debug exceptions. Controls routing of Debug exceptions, and defines the debug target Exception level, EL_D.

| TDE | Meaning |
|-----|--|
| 0b0 | The debug target Exception level is EL1. |
| 0b1 | If EL2 is enabled for the current Effective value of SCR .NS, the debug target Exception level is EL2, otherwise the debug target Exception level is EL1. The HDCR .{TDRA, TDOSA, TDA} fields are treated as being 1 for all purposes other than returning the result of a direct read of the register. |

For more information, see 'Routing debug exceptions'.

When [HCR](#).TGE == 1, the PE behaves as if the value of this field is 1 for all purposes other than returning the value of a direct read of the register.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

HPME, bit [7]

When FEAT_PMuV3 is implemented:

[[HDCR](#).HPMN..(N-1)] event counters enable.

| HPME | Meaning |
|------|---|
| 0b0 | Event counters in the range [HDCR.HPMN.. PMCR.N-1] are disabled. |
| 0b1 | Event counters in the range [HDCR.HPMN.. PMCR.N-1] are enabled by PMCNTENSET . |

If HDCR.HPMN is less than [PMCR.N](#), this field affects the operation of event counters in the range [HDCR.HPMN..[PMCR.N-1](#)].

This field does not affect the operation of other event counters.

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TPM, bit [6]

When FEAT_PMUv3 is implemented:

Trap Performance Monitors accesses. Traps Non-secure EL0 and EL1 accesses to all Performance Monitors registers to Hyp mode.

| TPM | Meaning |
|-----|--|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL0 and EL1 accesses to all Performance Monitors registers are trapped to Hyp mode. |

Note

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

TPMCR, bit [5]

When FEAT_PMUv3 is implemented:

Trap [PMCR](#) accesses. Traps Non-secure EL0 and EL1 accesses to the [PMCR](#) to Hyp mode.

| TPMCR | Meaning |
|-------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Non-secure EL0 and EL1 accesses to the PMCR are trapped to Hyp mode, unless it is trapped by PMUSERENR.EN . |

Note

EL2 does not provide traps on Performance Monitor register accesses through the optional memory-mapped external debug interface.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

HPMN, bits [4:0]**When FEAT_PMUv3 is implemented:**

Defines the number of event counters that are accessible from Non-secure EL1 modes, and from Non-secure EL0 modes if unprivileged access is enabled.

If HPMN is not 0 and is less than [PMCR.N](#), HPMN divides the event counters into a first range [0..(HPMN-1)], and a second range [HPMN..([PMCR.N](#)-1)].

If FEAT_HPMN0 is implemented and this field is 0, all event counters are in the second range and none are in the first range.

If HPMN is equal to [PMCR.N](#), all event counters are in the first range, and none are in the second range.

For an event counter <n> in the first range:

- The counter is accessible from EL1, EL2, EL3.
- The counter is accessible from EL0 if permitted by [PMUSERENR](#).
- If FEAT_PMUv3p5 is implemented, [PMCR.LP](#) determines whether the counter overflows at [PMEVCNTR<n>\[31:0\]](#) or [PMEVCNTR<n>\[63:0\]](#).
- [PMCR.E](#) and [PMCNTENSET\[n\]](#) enable the operation of event counter n.

For an event counter <n> in the second range:

- The counter is accessible from EL2 and EL3.
- If EL2 is disabled in the current Security state, the event counter is also accessible from EL1, and from EL0 if permitted by [PMUSERENR](#).
- If FEAT_PMUv3p5 is implemented, [HDCR.HLP](#) determines whether the counter overflows at [PMEVCNTR<n>\[31:0\]](#) or [PMEVCNTR<n>\[63:0\]](#).
- [HDCR.HPME](#) and [PMCNTENSET\[n\]](#) enable the operation of event counter n.

If HPMN is larger than [PMCR.N](#), or if FEAT_HPMN0 is not implemented and HPMN is 0, the following CONSTRAINED UNPREDICTABLE behaviors apply:

- The value returned by a direct read of [HDCR.HPMN](#) is UNKNOWN.
- Either:
 - An UNKNOWN number of counters are reserved for EL2 use. That is, the PE behaves as if [HDCR.HPMN](#) is set to an UNKNOWN non-zero value less than or equal to [PMCR.N](#).
 - All counters are reserved for EL2 use, meaning no counters are accessible from Non-secure EL1 and Non-secure EL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to the value in [PMCR.N](#).

Otherwise:

Reserved, RES0.

Accessing HDCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b001 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return HDCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HDCR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            HDCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HDCR = R[t];

```

HDFAR, Hyp Data Fault Address Register

The HDFAR characteristics are:

Purpose

Holds the virtual address of the faulting address that caused a synchronous Data Abort exception that is taken to Hyp mode.

Configuration

AArch32 System register HDFAR bits [31:0] are architecturally mapped to AArch64 System register [FAR_EL2\[31:0\]](#).

AArch32 System register HDFAR bits [31:0] are architecturally mapped to AArch32 System register [DFAR\[31:0\]](#) (S) when EL2 is implemented, EL3 is implemented and the implementation only supports execution in AArch32 state.

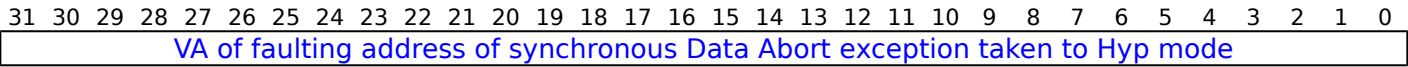
This register is present only when AArch32 is supported. Otherwise, direct accesses to HDFAR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HDFAR is a 32-bit register.

Field descriptions



Bits [31:0]

VA of faulting address of synchronous Data Abort exception taken to Hyp mode.

On a Prefetch Abort exception, this register is UNKNOWN.

Any execution in a Non-secure EL1 or Non-secure EL0 mode makes this register UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HDFAR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HDFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HDFAR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0110 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HDFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HDFAR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HIFAR, Hyp Instruction Fault Address Register

The HIFAR characteristics are:

Purpose

Holds the virtual address of the faulting address that caused a synchronous Prefetch Abort exception that is taken to Hyp mode.

Configuration

AArch32 System register HIFAR bits [31:0] are architecturally mapped to AArch64 System register [FAR_EL2\[63:32\]](#).

AArch32 System register HIFAR bits [31:0] are architecturally mapped to AArch32 System register [IFAR\[31:0\]](#) (S) when EL2 is implemented, EL3 is implemented and the implementation only supports execution in AArch32 state.

This register is present only when AArch32 is supported. Otherwise, direct accesses to HIFAR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HIFAR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA of faulting address of synchronous Prefetch Abort exception taken to Hyp mode | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

VA of faulting address of synchronous Prefetch Abort exception taken to Hyp mode.

On a Data Abort exception, this register is UNKNOWN.

Any execution in a Non-secure EL1 or Non-secure EL0 mode makes this register UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HIFAR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0110 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HIFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HIFAR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0110 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HIFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HIFAR = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HMAIR0, Hyp Memory Attribute Indirection Register 0

The HMAIR0 characteristics are:

Purpose

Along with [HMAIR1](#), provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations for memory accesses from Hyp mode.

AttrIndx[2] indicates the HMAIR register to be used:

- When AttrIndx[2] is 0, HMAIR0 is used.
- When AttrIndx[2] is 1, [HMAIR1](#) is used.

Configuration

AArch32 System register HMAIR0 bits [31:0] are architecturally mapped to AArch64 System register [MAIR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HMAIR0 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HMAIR0 is a 32-bit register.

Field descriptions

When TTBCR.EAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------|----|----|----|----|----|----|----|-----------------------|----|----|----|----|----|----|----|-----------------------|----|----|----|----|----|---|---|-----------------------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Attr3 | | | | | | | | Attr2 | | | | | | | | Attr1 | | | | | | | | Attr0 | | | | | | | |

Attr<n>, bits [8n+7:8n], for n = 3 to 0

The memory attribute encoding for an AttrIndx[2:0] entry in a Long descriptor format translation table entry, where:

- AttrIndx[2:0] gives the value of <n> in Attr<n>.
- AttrIndx[2] defines which MAIR to access. Attr7 to Attr4 are in MAIR1, and Attr3 to Attr0 are in MAIR0.

Bits [7:4] are encoded as follows:

| Attr<n>[7:4] | Meaning |
|---------------------|--|
| 0b0000 | Device memory. See encoding of Attr<n>[3:0] for the type of Device memory. |
| 0b00RW, RW not 0b00 | Normal memory, Outer Write-Through Transient. |
| 0b0100 | Normal memory, Outer Non-cacheable. |
| 0b01RW, RW not 0b00 | Normal memory, Outer Write-Back Transient. |
| 0b10RW | Normal memory, Outer Write-Through Non-transient. |
| 0b11RW | Normal memory, Outer Write-Back Non-transient. |

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

The meaning of bits [3:0] depends on the value of bits [7:4]:

| Attr<n>[3:0] | Meaning when Attr<n>[7:4] is 0b0000 | Meaning when Attr<n>[7:4] is not 0b0000 |
|------------------------|---|--|
| 0b0000 | Device-nGnRnE memory | UNPREDICTABLE |
| 0b00RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Through Transient |
| 0b0100 | Device-nGnRE memory | Normal memory, Inner Non-cacheable |
| 0b01RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Back Transient |
| 0b1000 | Device-nGRE memory | Normal memory, Inner Write-Through Non-transient (RW=0b00) |
| 0b10RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Through Non-transient |
| 0b1100 | Device-GRE memory | Normal memory, Inner Write-Back Non-transient (RW=0b00) |
| 0b11RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Back Non-transient |

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in some Attr<n> fields have the following meanings:

| R or W | Meaning |
|--------|-------------|
| 0b0 | No Allocate |
| 0b1 | Allocate |

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HMAIR0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HMAIR0;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HMAIR0;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HMAIR0 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HMAIR0 = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HMAIR1, Hyp Memory Attribute Indirection Register 1

The HMAIR1 characteristics are:

Purpose

Along with [HMAIR0](#), provides the memory attribute encodings corresponding to the possible AttrIndx values in a Long-descriptor format translation table entry for stage 1 translations for memory accesses from Hyp mode.

AttrIndx[2] indicates the HMAIR register to be used:

- When AttrIndx[2] is 0, [HMAIR0](#) is used.
- When AttrIndx[2] is 1, HMAIR1 is used.

Configuration

AArch32 System register HMAIR1 bits [31:0] are architecturally mapped to AArch64 System register [MAIR_EL2\[63:32\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HMAIR1 are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HMAIR1 is a 32-bit register.

Field descriptions

When TTBCR.EAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|-------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Attr7 | | | | | | | | Attr6 | | | | | | | | Attr5 | | | | | | | | Attr4 | | | | | | | |

Attr<n>, bits [8(n-4)+7:8(n-4)], for n = 7 to 4

The memory attribute encoding for an AttrIndx[2:0] entry in a Long descriptor format translation table entry, where:

- AttrIndx[2:0] gives the value of <n> in Attr<n>.
- AttrIndx[2] defines which MAIR to access. Attr7 to Attr4 are in MAIR1, and Attr3 to Attr0 are in MAIR0.

Bits [7:4] are encoded as follows:

| Attr<n>[7:4] | Meaning |
|---------------------|--|
| 0b0000 | Device memory. See encoding of Attr<n>[3:0] for the type of Device memory. |
| 0b00RW, RW not 0b00 | Normal memory, Outer Write-Through Transient. |
| 0b0100 | Normal memory, Outer Non-cacheable. |
| 0b01RW, RW not 0b00 | Normal memory, Outer Write-Back Transient. |
| 0b10RW | Normal memory, Outer Write-Through Non-transient. |
| 0b11RW | Normal memory, Outer Write-Back Non-transient. |

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

The meaning of bits [3:0] depends on the value of bits [7:4]:

| Attr<n>[3:0] | Meaning when Attr<n>[7:4] is 0b0000 | Meaning when Attr<n>[7:4] is not 0b0000 |
|------------------------|---|--|
| 0b0000 | Device-nGnRnE memory | UNPREDICTABLE |
| 0b00RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Through Transient |
| 0b0100 | Device-nGnRE memory | Normal memory, Inner Non-cacheable |
| 0b01RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Back Transient |
| 0b1000 | Device-nGRE memory | Normal memory, Inner Write-Through Non-transient (RW=0b00) |
| 0b10RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Through Non-transient |
| 0b1100 | Device-GRE memory | Normal memory, Inner Write-Back Non-transient (RW=0b00) |
| 0b11RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Back Non-transient |

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in some Attr<n> fields have the following meanings:

| R or W | Meaning |
|--------|-------------|
| 0b0 | No Allocate |
| 0b1 | Allocate |

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HMAIR1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1010 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HMAIR1;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HMAIR1;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1010 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HMAIR1 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HMAIR1 = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HPFAR, Hyp IPA Fault Address Register

The HPFAR characteristics are:

Purpose

Holds the faulting IPA for some aborts on a stage 2 translation taken to Hyp mode.

Configuration

AArch32 System register HPFAR bits [31:0] are architecturally mapped to AArch64 System register [HPFAR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HPFAR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HPFAR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| <div><div>FIPA[39:12]</div><div>RES0</div></div> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Execution in any Non-secure mode other than Hyp mode makes this register UNKNOWN.

FIPA[39:12], bits [31:4]

Bits [39:12] of the faulting intermediate physical address.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [3:0]

Reserved, RES0.

Accessing HPFAR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0110 | 0b0000 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HPFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HPFAR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0110 | 0b0000 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HPFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HPFAR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HRMR, Hyp Reset Management Register

The HRMR characteristics are:

Purpose

If EL2 is the highest implemented Exception level and this register is implemented:

- A write to the register at EL2 can request a Warm reset.
- If EL2 can use AArch32 and AArch64, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

AArch32 System register HRMR bits [31:0] are architecturally mapped to AArch64 System register [RMR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HRMR are UNDEFINED.

Only implemented if EL2 is the highest implemented Exception level. In this case:

- If EL2 can use AArch32 and AArch64 then this register must be implemented.
- If EL2 cannot use AArch64 then it is IMPLEMENTATION DEFINED whether the register is implemented.

Attributes

HRMR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | RR | | | | | | | | | | | | AA64 | | | |

Bits [31:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

AA64, bit [0]

When EL2 can use AArch64, determines which Execution state the PE boots into after a Warm reset:

| AA64 | Meaning |
|------|----------|
| 0b0 | AArch32. |
| 0b1 | AArch64. |

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If EL2 cannot use AArch64 this bit is RAZ/WI.

When implemented as a RW field, this field resets to 0 on a Cold reset.

Accessing HRMR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL1 && EL2Enabled() && IsHighestEL(EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12 ==
'1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && IsHighestEL(EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
    AArch32.TakeHypTrapException(0x03);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    return HRMR;
else
    UNDEFINED;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL1 && EL2Enabled() && IsHighestEL(EL2) && !ELUsingAArch32(EL2) && HSTR_EL2.T12 ==
'1' then
    AArch64.AArch32SystemAccessTrap(EL2, 0x03);
elsif PSTATE.EL == EL1 && EL2Enabled() && IsHighestEL(EL2) && ELUsingAArch32(EL2) && HSTR.T12 ==
'1' then
    AArch32.TakeHypTrapException(0x03);
elsif PSTATE.EL == EL2 && IsHighestEL(EL2) then
    HRMR = R[t];
else
    UNDEFINED;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HSCTLR, Hyp System Control Register

The HSCTLR characteristics are:

Purpose

Provides top level control of the system operation in Hyp mode.

Configuration

AArch32 System register HSCTLR bits [31:0] are architecturally mapped to AArch64 System register [SCTLR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HSCTLR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HSCTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|------|------|----|------|------|------|-----|------|------|------|------|----|------|------|-----|-----|------|------|-----|--------|---|----|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 |
| DSSBS | TE | RES1 | RES0 | EE | RES0 | RES1 | RES0 | WXN | RES1 | RES0 | RES1 | RES0 | I | RES1 | RES0 | SED | ITD | RES0 | CP15 | BEN | LSMAOE | h | TL | S | | | | |

DSSBS, bit [31]

When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry. The defined values are:

| DSSBS | Meaning |
|-------|--|
| 0b0 | PSTATE.SSBS is set to 0 on an exception to Hyp mode. |
| 0b1 | PSTATE.SSBS is set to 1 on an exception to Hyp mode. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

TE, bit [30]

T32 Exception Enable. This bit controls whether exceptions to EL2 are taken to A32 or T32 state:

| TE | Meaning |
|-----|--|
| 0b0 | Exceptions, including reset, taken to A32 state. |
| 0b1 | Exceptions, including reset, taken to T32 state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Bits [29:28]

Reserved, RES1.

Bits [27:26]

Reserved, RES0.

EE, bit [25]

The value of the PSTATE.E bit on entry to Hyp mode, the endianness of stage 1 translation table walks in the EL2 translation regime, and the endianness of stage 2 translation table walks in the PL1&0 translation regime.

| EE | Meaning |
|-----|---|
| 0b0 | Little-endian. PSTATE.E is cleared to 0 on entry to Hyp mode. Stage 1 translation table walks in the EL2 translation regime, and stage 2 translation table walks in the PL1&0 translation regime are little-endian. |
| 0b1 | Big-endian. PSTATE.E is set to 1 on entry to Hyp mode. Stage 1 translation table walks in the EL2 translation regime, and stage 2 translation table walks in the PL1&0 translation regime are big-endian. |

If an implementation does not provide Big-endian support at Exception levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support at Exception levels higher than EL0, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Bit [24]

Reserved, RES0.

Bits [23:22]

Reserved, RES1.

Bits [21:20]

Reserved, RES0.

WXN, bit [19]

Write permission implies XN (Execute-never). For the EL2 translation regime, this bit can force all memory regions that are writable to be treated as XN.

| WXN | Meaning |
|-----|--|
| 0b0 | This control has no effect on memory access permissions. |
| 0b1 | Any region that is writable in the EL2 translation regime is forced to XN for accesses from software executing at EL2. |

This bit applies only when HSCTLR.M bit is set.

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

Bit [18]

Reserved, RES1.

Bit [17]

Reserved, RES0.

Bit [16]

Reserved, RES1.

Bits [15:13]

Reserved, RES0.

I, bit [12]

Instruction access Cacheability control, for accesses at EL2:

| I | Meaning |
|-----|--|
| 0b0 | All instruction access to Normal memory from EL2 are Non-cacheable for all levels of instruction and unified cache. If the value of HSTCLR.M is 0, instruction accesses from stage 1 of the EL2 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory. |
| 0b1 | All instruction access to Normal memory from EL2 can be cached at all levels of instruction and unified cache. If the value of HSTCLR.M is 0, instruction accesses from stage 1 of the EL2 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory. |

This bit has no effect on the PL1&0 translation regime.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Bit [11]

Reserved, RES1.

Bits [10:9]

Reserved, RES0.

SED, bit [8]

SETEND instruction disable. Disables SETEND instructions at EL2.

| SED | Meaning |
|-----|---|
| 0b0 | SETEND instruction execution is enabled at EL2. |
| 0b1 | SETEND instructions are UNDEFINED at EL2. |

If the implementation does not support mixed-endian operation at EL2, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

ITD, bit [7]

IT Disable. Disables some uses of IT instructions at EL2.

| ITD | Meaning |
|-----|--|
| 0b0 | All IT instruction functionality is enabled at EL2. |
| 0b1 | Any attempt at EL2 to execute any of the following is UNDEFINED: <ul style="list-style-type: none"> All encodings of the IT instruction with hw1[3:0]≠1000. All encodings of the subsequent instruction with the following values for hw1: <ul style="list-style-type: none"> 11xxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM. 1011xxxxxxxxxxxx: All instructions in 'Miscellaneous 16-bit instructions'. 10100xxxxxxxxxxxx: ADD Rd, PC, #imm 01001xxxxxxxxxxxx: LDR Rd, [PC, #imm] 0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC. 010001xx1xxx111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers unpredictable cases with BLX Rn. <p>These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block.</p> <p>It is IMPLEMENTATION DEFINED whether the IT instruction is treated as:</p> <ul style="list-style-type: none"> A 16-bit instruction, that can only be followed by another 16-bit instruction. The first half of a 32-bit instruction. <p>This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED.</p> <p>An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.</p> |

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONSTRAINED UNPREDICTABLE. For more information, see 'Changes to an ITD control by an instruction in an IT block'.

ITD is optional, but if it is implemented in the HSCTLR then it must also be implemented in the [SCTLR_EL1](#), [SCTLR_EL2](#), and [SCTLR](#).

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement ITD, access to this field is **RAZ/WI**.

Bit [6]

Reserved, RES0.

CP15BEN, bit [5]

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from EL2:

| CP15BEN | Meaning |
|---------|---|
| 0b0 | EL2 execution of the CP15DMB , CP15DSB , and CP15ISB instructions is UNDEFINED. |
| 0b1 | EL2 execution of the CP15DMB , CP15DSB , and CP15ISB instructions is enabled. |

CP15BEN is optional, but if it is implemented in the HSCTLR then it must also be implemented in the [SCTLR_EL1](#), [SCTLR_EL2](#), and [SCTLR](#).

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

When an implementation does not implement CP15BEN, access to this field is **RAO/WI**.

LSMAOE, bit [4]

When FEAT_LSMAOC is implemented:

Load Multiple and Store Multiple Atomicity and Ordering Enable.

| LSMAOE | Meaning |
|--------|---|
| 0b0 | For all memory accesses at EL2, A32 and T32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered. |
| 0b1 | The ordering and interrupt behavior of A32 and T32 Load Multiple and Store Multiple at EL2 is as defined for Armv8.0. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 1.

Otherwise:

Reserved, RES1.

nTLSMD, bit [3]

When FEAT_LSMAOC is implemented:

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

| nTLSMD | Meaning |
|--------|--|
| 0b0 | All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL2 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault. |
| 0b1 | All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL2 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 1.

Otherwise:

Reserved, RES1.

C, bit [2]

Cacheability control, for data accesses at EL2:

| C | Meaning |
|----------|--|
| 0b0 | All data access to Normal memory from EL2, and all accesses to the EL2 translation tables, are Non-cacheable for all levels of data and unified cache. |
| 0b1 | All data access to Normal memory from EL2, and all accesses to the EL2 translation tables, can be cached at all levels of data and unified cache. |

This bit has no effect on the PL1&0 translation regime.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at EL2:

| A | Meaning |
|----------|---|
| 0b0 | Alignment fault checking disabled when executing at EL2. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element or data elements being accessed. |
| 0b1 | Alignment fault checking enabled when executing at EL2. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element or data elements being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception. |

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to an architecturally UNKNOWN value.

M, bit [0]

MMU enable for EL2 stage 1 address translation. Possible values of this bit are:

| M | Meaning |
|----------|---|
| 0b0 | EL2 stage 1 address translation disabled. See the HSCTLR.I field for the behavior of instruction accesses to Normal memory. |
| 0b1 | EL2 stage 1 address translation enabled. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2, this field resets to 0.

Accessing HSCTLR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b100 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HSCTLR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HSCTLR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HSCTLR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HSCTLR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HSR, Hyp Syndrome Register

The HSR characteristics are:

Purpose

Holds syndrome information for an exception taken to Hyp mode.

Configuration

AArch32 System register HSR bits [31:0] are architecturally mapped to AArch64 System register [ESR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HSR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EC | | | | | | IL | ISS | | | | | | | | | | | | | | | | | | | | | | | | |

Execution in any Non-secure PE mode other than Hyp mode makes this register UNKNOWN.

When an UNPREDICTABLE instruction is treated as UNDEFINED, and the exception is taken to EL2, the value of HSR is UNKNOWN. The value written to HSR must be consistent with a value that could be created as a result of an exception from the same Exception level that generated the exception as a result of a situation that is not UNPREDICTABLE at that Exception level, in order to avoid the possibility of a privilege violation.

EC, bits [31:26]

Exception Class. Indicates the reason for the exception that this register holds information about. Possible values of this field are:

| EC | Meaning | ISS |
|----------|---|---|
| 0b000000 | Unknown reason. | ISS encoding for exceptions with an unknown reason |
| 0b000001 | Trapped WFI or WFE instruction execution. Conditional WFE and WFI instructions that fail their condition code check do not cause an exception. | ISS encoding for Exception from a WFI or WFE instruction |
| 0b000011 | Trapped MCR or MRC access with (coproc==0b1111) that is not reported using EC 0b000000. | ISS encoding for Exception from an MCR or MRC access |
| 0b000100 | Trapped MCRR or MRRC access with (coproc==0b1111) that is not reported using EC 0b000000. | ISS encoding for Exception from an MCRR or MRRC access |
| 0b000101 | Trapped MCR or MRC access with (coproc==0b1110). | ISS encoding for Exception from an MCR or MRC access |
| 0b000110 | Trapped LDC or STC access. The only architected uses of these instructions are: <ul style="list-style-type: none"> An STC to write data to memory from DBGDTRRXint. An LDC to read data from memory to DBGDTRTXint. | ISS encoding for Exception from an LDC or STC instruction |
| 0b000111 | Access to Advanced SIMD or floating-point functionality trapped by a HCPTR .{TASE, TCP10} control. Excludes exceptions generated because Advanced SIMD and floating-point are not implemented. These are reported with EC value 0b000000. | ISS encoding for Exception from an access to SIMD or floating-point functionality, resulting from HCPTR |
| 0b001000 | Trapped VMRS access, from ID group trap, that is not reported using EC 0b000111. | ISS encoding for Exception from an MCR or MRC access |
| 0b001100 | Trapped MRRC access with (coproc==0b1110). | ISS encoding for Exception from an MCRR or MRRC access |
| 0b001110 | Illegal exception return to AArch32 state. | ISS encoding for Exception from an Illegal state or PC alignment fault |
| 0b010001 | Exception on SVC instruction execution in AArch32 state routed to EL2. | ISS encoding for Exception from HVC or SVC instruction execution |
| 0b010010 | HVC instruction execution in AArch32 state, when HVC is not disabled. | ISS encoding for Exception from HVC or SVC instruction execution |
| 0b010011 | Trapped execution of SMC instruction in AArch32 state. | ISS encoding for Exception from SMC instruction execution |
| 0b100000 | Prefetch Abort from a lower Exception level. | ISS encoding for Exception from a Prefetch Abort |
| 0b100001 | Prefetch Abort taken without a change in Exception level. | ISS encoding for Exception from a Prefetch Abort |
| 0b100010 | PC alignment fault exception. | ISS encoding for Exception from an |

| | | |
|----------|---|--|
| 0b100100 | Data Abort from a lower Exception level. | Illegal state or PC alignment fault |
| 0b100101 | Data Abort taken without a change in Exception level. | ISS encoding for Exception from a Data Abort |

All other EC values are reserved by Arm, and:

- Unused values in the range 0b000000 - 0b101100 (0x00 - 0x2C) are reserved for future use for synchronous exceptions.
- Unused values in the range 0b101101 - 0b111111 (0x2D - 0x3F) are reserved for future use, and might be used for synchronous or asynchronous exceptions.

The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IL, bit [25]

Instruction length bit. Indicates the size of the instruction that has been trapped to Hyp mode. When this bit is valid, possible values of this bit are:

| IL | Meaning |
|-----|-----------------------------|
| 0b0 | 16-bit instruction trapped. |
| 0b1 | 32-bit instruction trapped. |

This field is RES1 and not valid for the following cases:

- When the EC field is 0b000000, indicating an exception with an unknown reason.
- Prefetch Aborts.
- Data Aborts for which the HSR.ISS.ISV field is 0.
- When the EC value is 0b001110, indicating an Illegal state exception.

Note

This is a change from the behavior in Armv7, where the IL field is UNK/SBZP for the corresponding cases.

The IL field is not valid and is UNKNOWN on an exception from a PC alignment fault.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ISS, bits [24:0]

Instruction Specific Syndrome. Architecturally, this field can be defined independently for each defined Exception class. However, in practice, some ISS encodings are used for more than one Exception class.

ISS encoding for exceptions with an unknown reason

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [24:0]

Reserved, RES0.

This EC code is used for all exceptions that are not covered by any other EC value. This includes exceptions that are generated in the following situations:

- The attempted execution of an instruction bit pattern that has no allocated instruction or is not accessible in the current PE mode in the current Security state, including:
 - A read access using a System register encoding pattern that is not allocated for reads or that does not permit reads in the current PE mode and Security state.
 - A write access using a System register encoding pattern that is not allocated for writes or that does not permit writes in the current PE mode and Security state.
 - Instruction encodings that are unallocated.
 - Instruction encodings for instructions not implemented in the implementation.
- In Debug state, the attempted execution of an instruction bit pattern that is not accessible in Debug state.
- In Non-debug state, the attempted execution of an instruction bit pattern that is not accessible in Non-debug state.
- The attempted execution of a short vector floating-point instruction.
- In an implementation that does not include Advanced SIMD and floating-point functionality, an attempted access to Advanced SIMD or floating-point functionality under conditions where that access would be permitted if that functionality was present. This includes the attempted execution of an Advanced SIMD or floating-point instruction, and attempted accesses to Advanced SIMD and floating-point System registers.
- An exception generated because of the value of one of the [SCTLR](#).{ITD, SED, CP15BEN} control bits.
- Attempted execution of:
 - An HVC instruction when disabled by [HCR](#).HCD, [SCR](#).HCE, or [SCR_EL3](#).HCE.
 - An SMC instruction when disabled by [SCR](#).SCD or [SCR_EL3](#).SMD.
 - An HLT instruction when disabled by [EDSCR](#).HDE.
- An HVC instruction when disabled by [HCR](#).HCD, [SCR](#).HCE, or [SCR_EL3](#).HCE. An SMC instruction when disabled by [SCR](#).SCD or [SCR_EL3](#).SMD. An HLT instruction when disabled by [EDSCR](#).HDE.
- An exception generated because of the attempted execution of an MSR (Banked register) or MRS (Banked register) instruction that would access a Banked register that is not accessible from the Security state and PE mode at which the instruction was executed.

Note

An exception is generated only if the `CONSTRAINED UNPREDICTABLE` behavior of the instruction is that it is `UNDEFINED`, see 'MSR (banked register) and MRS (banked register)'.

- Attempted execution, in Debug state, of:
 - A DCPS1 instruction in Non-secure state from EL0 when EL2 is using AArch32 and the value of [HCR](#).TGE is 1.
 - A DCPS2 instruction at EL1 or EL0 when EL2 is not implemented, or when EL3 is using AArch32 and the value of [SCR](#).NS is 0, or when EL3 is using AArch64 and the value of [SCR_EL3](#).NS is 0.
 - A DCPS3 instruction when EL3 is not implemented, or when the value of [EDSCR](#).SDD is 1.
- In Debug state when the value of [EDSCR](#).SDD is 1, the attempted execution at EL2, EL1, or EL0 of an instruction that is configured to trap to EL3.

'Undefined Instruction exception, when the value of [HCR](#).TGE is 1' describes the configuration settings for a trap that returns an [HSR](#).EC value of 0b000000.

ISS encoding for Exception from a WFI or WFE instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|----|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | RES0 | | | | | | | | | | | | | | | | | | | TI |

CV, bit [24]

Condition code valid. Possible values of this bit are:

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is `IMPLEMENTATION DEFINED` whether CV is set to 1 or set to 0. For more information, see the description of the COND field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to 0b1110.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to 0b1110, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:1]

Reserved, RES0.

TI, bit [0]

Trapped instruction. Possible values of this bit are:

| TI | Meaning |
|-----|--------------|
| 0b0 | WFI trapped. |
| 0b1 | WFE trapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

'Traps to Hyp mode of Non-secure EL0 and EL1 execution of WFE and WFI instructions' describes the configuration settings for this trap.

ISS encoding for Exception from an MCR or MRC access

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|------|----|----|-----|----|----|------|----|---|---|-----|---|---|-----------|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | Opc2 | | | Opc1 | | | CRn | | | RES0 | Rt | | | CRm | | | Direction | | | |

CV, bit [24]

Condition code valid. Possible values of this bit are:

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. For more information, see the description of the COND field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to 0b1110.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to 0b1110, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc2, bits [19:17]

The Opc2 value from the issued instruction.

For a trapped VMRS access, holds the value 0b000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [16:14]

The Opc1 value from the issued instruction.

For a trapped VMRS access, holds the value 0b111.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRn, bits [13:10]

The CRn value from the issued instruction.

For a trapped VMRS access, holds the reg field from the VMRS instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

Rt, bits [8:5]

The Rt value from the issued instruction, the general-purpose register used for the transfer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

For a trapped VMRS access, holds the value 0b0000.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|---|
| 0b0 | Write to System register space. MCR instruction. |
| 0b1 | Read from System register space. MRC or VMRS instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following sections describe configuration settings for traps that are reported using EC value 0b000011:

- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to the ID registers'.
- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to lockdown, DMA, and TCM operations'.
- 'Traps to Hyp mode of Non-secure EL1 execution of cache maintenance instructions'.
- 'Traps to Hyp mode of Non-secure EL1 execution of TLB maintenance instructions'.
- 'Traps to Hyp mode of Non-secure EL1 accesses to the Auxiliary Control Register'.
- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to Performance Monitors registers'.
- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to Activity Monitors registers'.
- 'Traps to Hyp mode of Non-secure EL1 accesses to the CPACR'.
- 'Traps to Hyp mode of Non-secure EL1 accesses to virtual memory control registers'.
- 'General trapping to Hyp mode of Non-secure EL0 and EL1 accesses to System registers in the (coproc == 1111) encoding space'.

The following sections describe configuration settings for traps that are reported using EC value 0b000101:

- 'ID group 0, Primary device identification registers'.
- 'Traps to Hyp mode of Non-secure System register accesses to trace registers'.
- 'Trapping Non-secure System register accesses to Debug ROM registers'.
- 'Trapping Non-secure System register accesses to powerdown debug registers'.
- 'Trapping general Non-secure System register accesses to debug registers'.

The following sections describes configuration settings for traps that are reported using EC value 0b001000:

- 'ID group 0, Primary device identification registers'.
- 'ID group 3, Detailed feature identification registers'.

ISS encoding for Exception from an MCRR or MRRC access

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|------|-----|----|----|----|------|----|---|---|---|-----|---|---|---|-----------|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | Opc1 | | | | RES0 | Rt2 | | | | RES0 | Rt | | | | CRm | | | | Direction | |

CV, bit [24]

Condition code valid. Possible values of this bit are:

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. For more information, see the description of the COND field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to 0b1110.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to 0b1110, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Opc1, bits [19:16]

The Opc1 value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:14]

Reserved, RES0.

Rt2, bits [13:10]

The Rt2 value from the issued instruction, the second general-purpose register used for the transfer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [9]

Reserved, RES0.

Rt, bits [8:5]

The Rt value from the issued instruction, the first general-purpose register used for the transfer.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CRm, bits [4:1]

The CRm value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|--|
| 0b0 | Write to System register space. MCRR instruction. |
| 0b1 | Read from System register space. MRRC instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following sections describe configuration settings for traps that are reported using EC value 0b000100:

- 'Traps to Hyp mode of Non-secure EL1 accesses to virtual memory control registers'.
- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to Performance Monitors registers'.
- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to Activity Monitors registers'.
- 'Traps to Hyp mode of Non-secure EL0 and EL1 accesses to the Generic Timer registers'.
- 'General trapping to Hyp mode of Non-secure EL0 and EL1 accesses to System registers in the (coproc == 1111) encoding space'.

The following sections describe configuration settings for traps that are reported using EC value 0b001100:

- 'Traps to Hyp mode of Non-secure System register accesses to trace registers'.
- 'Trapping Non-secure System register accesses to Debug ROM registers'.

ISS encoding for Exception from an LDC or STC instruction

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|----|----|----|----|------|----|---|----|---|---|--------|----|---|-----------|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | imm8 | | | | | | | | RES0 | | | Rn | | | Offset | AM | | Direction | | |

CV, bit [24]

Condition code valid. Possible values of this bit are:

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. For more information, see the description of the COND field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to 0b1110.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to 0b1110, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

imm8, bits [19:12]

The immediate value from the issued instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:9]

Reserved, RES0.

Rn, bits [8:5]

The Rn value from the issued instruction. Valid only when AM[2] is 0, indicating an immediate form of the LDC or STC instruction.

When AM[2] is 1, indicating a literal form of the LDC or STC instruction, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Offset, bit [4]

Indicates whether the offset is added or subtracted:

| Offset | Meaning |
|--------|------------------|
| 0b0 | Subtract offset. |
| 0b1 | Add offset. |

This bit corresponds to the U bit in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AM, bits [3:1]

Addressing mode. The permitted values of this field are:

| AM | Meaning |
|-------|---|
| 0b000 | Immediate unindexed. |
| 0b001 | Immediate post-indexed. |
| 0b010 | Immediate offset. |
| 0b011 | Immediate pre-indexed. |
| 0b100 | Literal unindexed. LDC instruction in A32 instruction set only. For a trapped STC instruction or a trapped T32 LDC instruction this encoding is reserved. |
| 0b110 | Literal offset. LDC instruction only. For a trapped STC instruction, this encoding is reserved. |

The values 0b101 and 0b111 are reserved. The effect of programming this field to a reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

Bit [2] in this subfield indicates the instruction form, immediate or literal.

Bits [1:0] in this subfield correspond to the bits {P, W} in the instruction encoding.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Direction, bit [0]

Indicates the direction of the trapped instruction.

| Direction | Meaning |
|-----------|------------------------------------|
| 0b0 | Write to memory. STC instruction. |
| 0b1 | Read from memory. LDC instruction. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

'Trapping general Non-secure System register accesses to debug registers' describes the configuration settings for the trap that is reported using EC value 0b000110.

ISS encoding for Exception from an access to SIMD or floating-point functionality, resulting from HCPTR

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|------|----|----|----|----|----|----|----|----|----|---------|---|--------|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | RES0 | | | | | | | | | | TA RES0 | | coproc | | | | | | | |

Excludes exceptions that occur because Advanced SIMD and floating-point functionality is not implemented, or because the value of [HCR.TGE](#) or [HCR_EL2.TGE](#) is 1. These are reported with EC value 0b000000.

CV, bit [24]

Condition code valid. Possible values of this bit are:

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. For more information, see the description of the COND field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to 0b1110.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to 0b1110, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [19:6]

Reserved, RES0.

TA, bit [5]

Indicates trapped use of Advanced SIMD functionality.

| TA | Meaning |
|-----|---|
| 0b0 | Exception was not caused by trapped use of Advanced SIMD functionality. |
| 0b1 | Exception was caused by trapped use of Advanced SIMD functionality. |

Any use of an Advanced SIMD instruction that is not also a floating-point instruction that is trapped to Hyp mode because of a trap configured in the [HCPTR](#) sets this bit to 1.

For a list of these instructions, see 'Controls of Advanced SIMD operation that do not apply to floating-point operation'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [4]

Reserved, RES0.

coproc, bits [3:0]

When the [HSR.TA](#) field returns the value 1, this field returns the value 0b1010. Otherwise, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following sections describe the configuration settings for the traps that are reported using EC value 0b000111:

- 'General trapping to Hyp mode of Non-secure accesses to the SIMD and floating-point registers'.
- 'Traps to Hyp mode of Non-secure accesses to Advanced SIMD functionality'.

ISS encoding for Exception from HVC or SVC instruction execution

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | imm16 | | | | | | | | | | | | | | |

Bits [24:16]

Reserved, RES0.

imm16, bits [15:0]

The value of the immediate field from the HVC or SVC instruction.

For an HVC instruction, this is the value of the imm16 field of the issued instruction.

For an SVC instruction:

- If the instruction is unconditional, then:
 - For the T32 instruction, this field is zero-extended from the imm8 field of the instruction.
 - For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- For the T32 instruction, this field is zero-extended from the imm8 field of the instruction. For the A32 instruction, this field is the bottom 16 bits of the imm24 field of the instruction.
- If the instruction is conditional, this field is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The HVC instruction is unconditional, and a conditional SVC instruction generates an exception only if it passes its condition code check. Therefore, the syndrome information for these exceptions does not require conditionality information.

'Supervisor Call exception, when the value of HCR.TGE is 1' describes the configuration settings for the trap reported with EC value 0b010001.

ISS encoding for Exception from SMC instruction execution

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|-------------|------|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CV | COND | | | | CCKNOWNPASS | RES0 | | | | | | | | | | | | | | | | | | |

CV, bit [24]

Condition code valid. Possible values of this bit are:

| CV | Meaning |
|-----|------------------------------|
| 0b0 | The COND field is not valid. |
| 0b1 | The COND field is valid. |

When an A32 instruction is trapped, CV is set to 1.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether CV is set to 1 or set to 0. For more information, see the description of the COND field.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

COND, bits [23:20]

The condition code for the trapped instruction.

When an A32 instruction is trapped, CV is set to 1 and:

- If the instruction is conditional, COND is set to the condition code field value from the instruction.
- If the instruction is unconditional, COND is set to 0b1110.

A conditional A32 instruction that is known to pass its condition code check can be presented either:

- With COND set to 0b1110, the value for unconditional.
- With the COND value held in the instruction.

When a T32 instruction is trapped, it is IMPLEMENTATION DEFINED whether:

- CV is set to 0 and COND is set to an UNKNOWN value. Software must examine the SPSR.IT field to determine the condition, if any, of the T32 instruction.
- CV is set to 1 and COND is set to the condition code for the condition that applied to the instruction.

For an implementation that, for both A32 and T32 instructions, takes an exception on a trapped conditional instruction only if the instruction passes its condition code check, these definitions mean that when CV is set to 1 it is IMPLEMENTATION DEFINED whether the COND field is set to 0b1110, or to the value of any condition that applied to the instruction.

This field is valid only if CCKNOWNPASS is 1, otherwise it is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CCKNOWNPASS, bit [19]

Indicates whether the instruction might have failed its condition code check.

| CCKNOWNPASS | Meaning |
|-------------|--|
| 0b0 | The instruction was unconditional, or was conditional and passed its condition code check. |
| 0b1 | The instruction was conditional, and might have failed its condition code check. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [18:0]

Reserved, RES0.

'Traps to Hyp mode of Non-secure EL1 execution of SMC instructions' describes the configuration settings for this trap, for instructions executed in Non-secure EL1.

ISS encoding for Exception from a Prefetch Abort

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|------|-------|------|------|---|---|---|---|---|--|--|
| 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
| RES0 | | | | | | | | | | | | | | FnV | EA | RES0 | S1PTW | RES0 | IFSC | | | | | | | |

Bits [24:11]

Reserved, RES0.

FnV, bit [10]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|---|
| 0b0 | HIFAR is valid. |
| 0b1 | HIFAR is not valid, and holds an UNKNOWN value. |

This field is valid only if the IFSC code is 0b010000. It is RES0 for all other aborts.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [8]

Reserved, RES0.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

| S1PTW | Meaning |
|--------------|---|
| 0b0 | Fault not on a stage 2 translation for a stage 1 translation table walk. |
| 0b1 | Fault on the stage 2 translation of an access for a stage 1 translation table walk. |

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [6]

Reserved, RES0.

IFSC, bits [5:0]

Instruction Fault Status Code. Possible values of this field are:

| IFSC | Meaning | Applies when |
|-------------|--|----------------------------------|
| 0b000000 | Address size fault in translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010000 | Synchronous External abort, not on translation table walk. | |
| 0b010101 | Synchronous External abort on translation table walk, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk, level 3. | |
| 0b011000 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b011101 | Synchronous parity or ECC error on memory access on translation table walk, level 1. | When FEAT_RAS is not implemented |
| 0b011110 | Synchronous parity or ECC error on memory access on translation table walk, level 2. | When FEAT_RAS is not implemented |
| 0b011111 | Synchronous parity or ECC error on memory access on translation table walk, level 3. | When FEAT_RAS is not implemented |
| 0b100010 | Debug exception. | |
| 0b110000 | TLB conflict abort. | |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Long-descriptor translation table lookup'.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

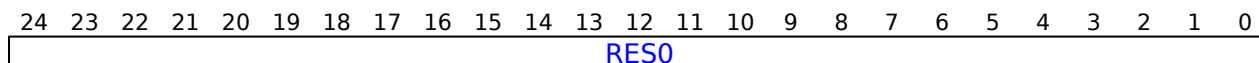
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following sections describe cases where Prefetch Abort exceptions can be routed to Hyp mode, generating exceptions that are reported in the HSR with EC value 0b100000:

- 'Abort exceptions, when the value of HCR.TGE is 1'.
- 'Routing debug exceptions to EL2 using AArch32'.

ISS encoding for Exception from an Illegal state or PC alignment fault



Bits [24:0]

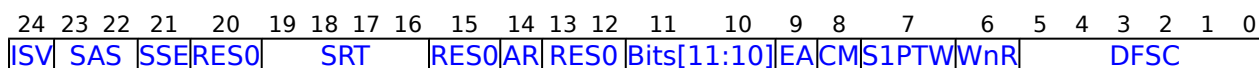
Reserved, RES0.

For more information about the Illegal state exception, see:

- 'Illegal changes to PSTATE.M'.
- 'Illegal return events from AArch32 state'.
- 'Legal returns that set PSTATE.IL to 1'.
- 'The Illegal Execution state exception'.

For more information about the PC alignment fault exception, see 'Branching to an unaligned PC'.

ISS encoding for Exception from a Data Abort



ISV, bit [24]

Instruction syndrome valid. Indicates whether the syndrome information in ISS[23:14] is valid.

| ISV | Meaning |
|-----|---|
| 0b0 | No valid instruction syndrome. ISS[23:14] are RES0. |
| 0b1 | ISS[23:14] hold a valid instruction syndrome. |

This bit is 0 for all faults except Data Aborts generated by stage 2 address translations for which all the following apply to the instruction that generated the Data Abort exception:

- The instruction is an LDR, LDA, LDRT, LDRSH, LDRSHT, LDRH, LDAH, LDRHT, LDRSB, LDRSBT, LDRB, LDAB, LDRBT, STR, STL, STRT, STRH, STLH, STRHT, STRB, STLB, or STRBT instruction.
- The instruction is not performing register writeback.
- The instruction is not using the PC as a source or destination register.

For these cases, ISV is UNKNOWN if the exception was generated in Debug state in memory access mode, as described in 'Data Aborts in Memory access mode', and otherwise indicates whether ISS[23:14] hold a valid syndrome.

Note

In the A32 instruction set, LDR*T and STR*T instructions always perform register writeback and therefore never return a valid instruction syndrome.

When FEAT_RAS is implemented, ISV is 0 for any synchronous External abort.

ISV is set to 0 on a stage 2 abort on a stage 1 translation table walk.

When FEAT_RAS is not implemented, it is IMPLEMENTATION DEFINED whether ISV is set to 1 or 0 on a synchronous External abort on a stage 2 translation table walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SAS, bits [23:22]

Syndrome Access Size. When ISV is 1, indicates the size of the access attempted by the faulting operation.

| SAS | Meaning |
|------|------------|
| 0b00 | Byte |
| 0b01 | Halfword |
| 0b10 | Word |
| 0b11 | Doubleword |

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SSE, bit [21]

Syndrome Sign Extend. When ISV is 1, for a byte, halfword, or word load operation, indicates whether the data item must be sign extended. For these cases, the possible values of this bit are:

| SSE | Meaning |
|-----|----------------------------------|
| 0b0 | Sign-extension not required. |
| 0b1 | Data item must be sign-extended. |

For all other operations this bit is 0.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [20]

Reserved, RES0.

SRT, bits [19:16]

Syndrome Register transfer. When ISV is 1, the register number of the Rt operand of the faulting instruction.

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [15]

Reserved, RES0.

AR, bit [14]

Acquire/Release. When ISV is 1, the possible values of this bit are:

| AR | Meaning |
|-----|---|
| 0b0 | Instruction did not have acquire/release semantics. |
| 0b1 | Instruction did have acquire/release semantics. |

This field is UNKNOWN when the value of ISV is UNKNOWN.

This field is RES0 when the value of ISV is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [13:12]

Reserved, RES0.

AET, bits [11:10]

When FEAT_RAS is implemented:

Asynchronous Error Type. When DFSC is 0b010001, describes the PE error state after taking the SError interrupt exception.

| AET | Meaning |
|------|----------------------------|
| 0b00 | Uncontainable (UC). |
| 0b01 | Unrecoverable state (UEU). |
| 0b10 | Restartable state (UEO). |
| 0b11 | Recoverable state (UER). |

On a synchronous Data Abort, this field is RES0.

In the event of multiple errors taken as a single SError interrupt exception, the overall PE error state is reported.

Note

Software can use this information to determine what recovery might be possible. The recovery software must also examine any implemented fault records to determine the location and extent of the error.

When FEAT_RAS is not implemented, or when DFSC is not 0b010001:

- Bit[11] is RES0.
- Bit[10] forms the FnV field.

Note

Armv8.2 requires the implementation of FEAT_RAS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|--|
| 0b0 | HDFAR is valid. |
| 0b1 | HDFAR is not valid, and holds an UNKNOWN value. |

When FEAT_RAS is not implemented, this field is valid only if DFSC is 0b010000. It is RES0 for all other aborts.

When FEAT_RAS is implemented:

- If DFSC is 0b010000, this field is valid.
- If DFSC is 0b010001, this bit forms part of the AET field, becoming AET[0].
- This field is RES0 for all other aborts.

Note

Armv8.2 requires the implementation of FEAT_RAS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EA, bit [9]

External abort type. This bit can provide an IMPLEMENTATION DEFINED classification of External aborts.

For any abort other than an External abort this bit returns a value of 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CM, bit [8]

Cache maintenance. For a synchronous fault, identifies fault that comes from a cache maintenance or address translation instruction. For synchronous faults, the possible values of this bit are:

| CM | Meaning |
|-----|--|
| 0b0 | Fault not generated by a cache maintenance or address translation instruction. |
| 0b1 | Fault generated by a cache maintenance or address translation instruction. |

For an asynchronous Data Abort exception, this bit is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S1PTW, bit [7]

For a stage 2 fault, indicates whether the fault was a stage 2 fault on an access made for a stage 1 translation table walk:

| S1PTW | Meaning |
|-------|---|
| 0b0 | Fault not on a stage 2 translation for a stage 1 translation table walk. |
| 0b1 | Fault on the stage 2 translation of an access for a stage 1 translation table walk. |

For any abort other than a stage 2 fault this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

WnR, bit [6]

Write not Read. Indicates whether a synchronous abort was caused by a write instruction or a read instruction.

| WnR | Meaning |
|------------|--------------------------------------|
| 0b0 | Abort caused by a read instruction. |
| 0b1 | Abort caused by a write instruction. |

For faults on cache maintenance and address translation instructions, this bit always returns a value of 1.

On an asynchronous Data Abort:

- When FEAT_RAS is not implemented, this bit is UNKNOWN.
- When FEAT_RAS is implemented, this bit is RES0.

Note

Armv8.2 requires the implementation of FEAT_RAS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DFSC, bits [5:0]

Data Fault Status Code. Possible values of this field are:

| DFSC | Meaning | Applies when |
|----------|--|----------------------------------|
| 0b000000 | Address size fault in translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010000 | Synchronous External abort, not on translation table walk. | |
| 0b010001 | Asynchronous SError interrupt. | |
| 0b010101 | Synchronous External abort on translation table walk, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk, level 3. | |
| 0b011000 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b011001 | Asynchronous SError interrupt, from a parity or ECC error on memory access. | When FEAT_RAS is not implemented |
| 0b011101 | Synchronous parity or ECC error on memory access on translation table walk, level 1. | When FEAT_RAS is not implemented |
| 0b011110 | Synchronous parity or ECC error on memory access on translation table walk, level 2. | When FEAT_RAS is not implemented |
| 0b011111 | Synchronous parity or ECC error on memory access on translation table walk, level 3. | When FEAT_RAS is not implemented |
| 0b100001 | Alignment fault. | |
| 0b100010 | Debug exception. | |
| 0b110000 | TLB conflict abort. | |
| 0b110100 | IMPLEMENTATION DEFINED fault (Lockdown). | |
| 0b110101 | IMPLEMENTATION DEFINED fault (Unsupported Exclusive access). | |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Long-descriptor translation table lookup'.

If the S1PTW bit is set, then the level refers the level of the stage2 translation that is translating a stage 1 translation walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The following describe cases where Data Abort exceptions can be routed to Hyp mode, generating exceptions that are reported in the HSR with EC value 0b100100:

- 'Abort exceptions, when the value of HCR.TGE is 1'.
- 'Routing debug exceptions to EL2 using AArch32'.

The following describe cases that can cause a Data Abort exception that is taken to Hyp mode, and reported in the HSR with EC value of 0b100000 or 0b100100:

- 'Hyp mode control of Non-secure access permissions'.
- 'Memory fault reporting in Hyp mode'.

Accessing HSR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HSR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0101 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HSR = R[t];

```

HSTR, Hyp System Trap Register

The HSTR characteristics are:

Purpose

Controls trapping to Hyp mode of Non-secure accesses, at EL1 or lower, to System registers in the coproc == 0b1111 encoding space:

- By the CRn value used to access the register using MCR or MRC instruction.
- By the CRm value used to access the register using MCRR or MRRC instruction.

Configuration

AArch32 System register HSTR bits [31:0] are architecturally mapped to AArch64 System register [HSTR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HSTR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HSTR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|------|-----|-----|-----|-----|----|----|----|----|----|------|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | T15 | RES0 | T13 | T12 | T11 | T10 | T9 | T8 | T7 | T6 | T5 | RES0 | T3 | T2 | T1 | T0 |

Bits [31:16, 14, 4]

Reserved, RES0.

T<n>, bit [n], for n = 15, 13 to 5, 3 to 0

The remaining fields control whether Non-secure EL0 and EL1 accesses, using MCR, MRC, MCRR, and MRRC instructions, to the System registers in the coproc == 0b1111 encoding space are trapped to Hyp mode:

| T<n> | Meaning |
|------|--|
| 0b0 | This control has no effect on Non-secure EL0 or EL1 accesses to System registers. |
| 0b1 | Any Non-secure EL1 MCR or MRC access with coproc == 0b1111 and CRn == <n> is trapped to Hyp mode. A Non-secure EL0 MCR or MRC access with these values is trapped to Hyp mode only if the access is not UNDEFINED when the value of this field is 0. Any Non-secure EL1 MCRR or MRRC access with coproc == 0b1111 and CRm == <n> is trapped to Hyp mode. A Non-secure EL0 MCRR or MRRC access with these values is trapped to Hyp mode only if the access is not UNDEFINED when the value of this field is 0. |

For example, when HSTR.T7 is 1, for instructions executed at Non-secure EL1:

- An MCR or MRC instruction with coproc set to 0b1111 and <CRn> set to c7 is trapped to Hyp mode.
- An MCRR or MRRC instruction with coproc set to 0b1111 and <CRm> set to c7 is trapped to Hyp mode.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Accessing HSTR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HSTR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HSTR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HSTR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HSTR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HTCR, Hyp Translation Control Register

The HTCR characteristics are:

Purpose

The control register for stage 1 of the EL2 translation regime.

Note

This stage of translation always uses the Long-descriptor translation table format.

Configuration

AArch32 System register HTCR bits [31:0] are architecturally mapped to AArch64 System register [TCR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HTCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HTCR is a 32-bit register.

Field descriptions

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 |
|------|---------------------------|------|-------|-------|-------|-------|-----|------|------|----|----|----|-----|-------|-------|------|-----|----|----|----|----|---|---|---|---|---|---|---|---|---|
| RES1 | IMPLEMENTATION DEFINED | RES0 | HWU62 | HWU61 | HWU60 | HWU59 | HPD | RES1 | RES0 | | | | SH0 | ORGN0 | IRGN0 | RES0 | T0S | | | | | | | | | | | | | |

Bit [31]

Reserved, RES1.

IMPLEMENTATION DEFINED, bit [30]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [29]

Reserved, RES0.

HWU62, bit [28]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry.

| HWU62 | Meaning |
|-------|--|
| 0b0 | Bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of HTCR.HPD is 1. |

The Effective value of this field is 0 if the value of HTCR.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU61, bit [27]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry.

| HWU61 | Meaning |
|-------|--|
| 0b0 | Bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of HTCR.HPD is 1. |

The Effective value of this field is 0 if the value of HTCR.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU60, bit [26]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry.

| HWU60 | Meaning |
|-------|--|
| 0b0 | Bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of HTCR.HPD is 1. |

The Effective value of this field is 0 if the value of HTCR.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU59, bit [25]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry.

| HWU59 | Meaning |
|-------|--|
| 0b0 | Bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of HTCR.HPD is 1. |

The Effective value of this field is 0 if the value of HTCR.HPD is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HPD, bit [24]**When FEAT_AA32HPD is implemented:**

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, XNTable, and PXNTable, in the PL2 translation regime.

| HPD | Meaning |
|-----|--|
| 0b0 | Hierarchical permissions are enabled. |
| 0b1 | Hierarchical permissions are disabled. |

When disabled, the permissions are treated as if the bits are zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES1.

Bits [22:14]

Reserved, RES0.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using [HTTBR](#).

| SH0 | Meaning |
|------|------------------|
| 0b00 | Non-shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using [HTTBR](#).

| ORGN0 | Meaning |
|-------|---|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using [HTTBR](#).

| IRGN0 | Meaning |
|-------|---|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [7:3]

Reserved, RES0.

T0SZ, bits [2:0]

The size offset of the memory region addressed by [HTTBR](#). The region size is $2^{(32-T0SZ)}$ bytes.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HTCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0010 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HTCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HTCR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0010 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HTCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HTCR = R[t];

```

HTPIDR, Hyp Software Thread ID Register

The HTPIDR characteristics are:

Purpose

Provides a location where software running in Hyp mode can store thread identifying information that is not visible to Non-secure software executing at EL0 or EL1, for hypervisor management purposes.

The PE makes no use of this register.

Configuration

AArch32 System register HTPIDR bits [31:0] are architecturally mapped to AArch64 System register [TPIDR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HTPIDR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

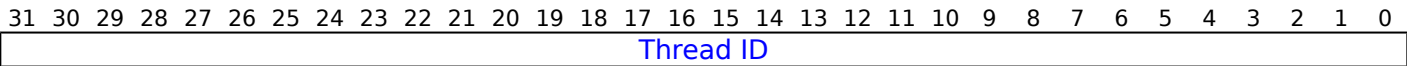
Note

The PE never updates this register.

Attributes

HTPIDR is a 32-bit register.

Field descriptions



Bits [31:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing HTPIDR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1101 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HTPIDR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HTPIDR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1101 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HTPIDR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HTPIDR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HTRFCR, Hyp Trace Filter Control Register

The HTRFCR characteristics are:

Purpose

Provides EL2 controls for Trace.

Configuration

AArch32 System register HTRFCR bits [31:0] are architecturally mapped to AArch64 System register [TRFCR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_TRF is implemented. Otherwise, direct accesses to HTRFCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from Monitor mode when [SCR.NS](#) == 1.

Attributes

HTRFCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|------|-------|--------|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | TS | RES0 | CX | RES0 | E2TRE | E0HTRE | | | | | | | | | | | | |

Bits [31:7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control. Controls which timebase is used for trace timestamps.

| TS | Meaning |
|------|--|
| 0b00 | The timestamp is controlled by TRFCR.TS . |
| 0b01 | Virtual timestamp. The traced timestamp is the physical counter value minus the value of CNTVOFF . |
| 0b11 | Physical timestamp. The traced timestamp is the physical counter value. |

When SelfHostedTraceEnabled() == FALSE, this field is ignored.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Bit [4]

Reserved, RES0.

CX, bit [3]

VMID Trace Enable.

| CX | Meaning |
|-----|------------------------------|
| 0b0 | VMID tracing is not allowed. |
| 0b1 | VMID tracing is allowed. |

When SelfHostedTraceEnabled() == FALSE, this field is ignored.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Bit [2]

Reserved, RES0.

E2TRE, bit [1]

EL2 Trace Enable.

| E2TRE | Meaning |
|-------|-------------------------------|
| 0b0 | Tracing is prohibited at EL2. |
| 0b1 | Tracing is allowed at EL2. |

When SelfHostedTraceEnabled() == FALSE, this field is ignored.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

EOHTRE, bit [0]

EL0 Trace Enable.

| EOHTRE | Meaning |
|--------|---|
| 0b0 | Tracing is prohibited at EL0 when HCR.TGE == 1. |
| 0b1 | Tracing is allowed at EL0 when HCR.TGE == 1. |

This field is ignored if any of the following are true:

- The PE is in Secure state.
- SelfHostedTraceEnabled() == FALSE.
- [HCR.TGE](#) == 0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

Accessing HTRFCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return HTRFCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HTRFCR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0001 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        HTRFCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HTRFCR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HTTBR, Hyp Translation Table Base Register

The HTTBR characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of an address translation in the EL2 translation regime, and other information for this translation regime.

Configuration

AArch32 System register HTTBR bits [47:1] are architecturally mapped to AArch64 System register [TTBR0_EL2\[47:1\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HTTBR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HTTBR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | BADDR | | | | | | | | | | | | | | | | |
| BADDR | | | | | | | | | | | | | | | | CnP | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:48]

Reserved, RES0.

BADDR, bits [47:1]

Translation table base address, bits[47:x], Bits [x-1:1] are RES0, with the additional requirement that if bits[x-1:3] are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:3] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

x is determined from the value of [HTCR.T0SZ](#) as follows:

- If [HTCR.T0SZ](#) is 0 or 1, $x = 5 - \text{HTCR.T0SZ}$.
- If [HTCR.T0SZ](#) is greater than 1, $x = 14 - \text{HTCR.T0SZ}$.

If bits[47:40] of the translation table base address are not zero, an Address size fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When [FEAT_TTCNP](#) is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by HTTBR is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of HTTBR.CnP is 1.

| CnP | Meaning |
|-----|--|
| 0b0 | The translation table entries pointed to by HTTBR are permitted to differ from corresponding entries for HTTBR for other PEs in the Inner Shareable domain. This is not affected by the value of HTTBR.CnP on those other PEs. |
| 0b1 | The translation table entries pointed to by HTTBR are the same as the translation table entries pointed to by HTTBR on every other PE in the Inner Shareable domain for which the value of HTTBR.CnP is 1. |

Note

If the value of the HTTBR.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those HTTBRs do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are **CONSTRAINED UNPREDICTABLE**, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing HTTBR

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b0010 | 0b0100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HTTBR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HTTBR;

```

MCRR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b0010 | 0b0100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HTTBR = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HTTBR = R[t2]:R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

HVBAR, Hyp Vector Base Address Register

The HVBAR characteristics are:

Purpose

Holds the vector base address for any exception that is taken to Hyp mode.

Configuration

AArch32 System register HVBAR bits [31:0] are architecturally mapped to AArch64 System register [VBAR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to HVBAR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

HVBAR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Vector Base Address | | | | | | | | | | | | | | | | | | | | | | | | | | | RES0 | | | | |

Bits [31:5]

Vector Base Address. Bits[31:5] of the base address of the exception vectors for exceptions taken to this Exception level. Bits[4:0] of an exception vector are the exception offset.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:0]

Reserved, RES0.

Accessing HVBAR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return HVBAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return HVBAR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    HVBAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        HVBAR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_AP0R<n>, Interrupt Controller Active Priorities Group 0 Registers, n = 0 - 3

The ICC_AP0R<n> characteristics are:

Purpose

Provides information about Group 0 active priorities.

Configuration

AArch32 System register ICC_AP0R<n> bits [31:0] are architecturally mapped to AArch64 System register [ICC_AP0R<n>_EL1\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_AP0R<n> are UNDEFINED.

Attributes

ICC_AP0R<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

Accessing ICC_AP0R<n>

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 0 active priorities) might result in UNPREDICTABLE behavior of the interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICC_AP0R1 is only implemented in implementations that support 6 or more bits of preemption. ICC_AP0R2 and ICC_AP0R3 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by [ICH_VTR](#).PREbits.

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICC_AP0R<n>.

- Secure [ICC_AP1R<n>](#).
- Non-secure [ICC_AP1R<n>](#).

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b1:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICC_AP0R[UInt(opc2<1:0>)];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICC_AP0R[UInt(opc2<1:0>)];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_AP0R[UInt(opc2<1:0>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_AP0R[UInt(opc2<1:0>)];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            return ICC_AP0R[UInt(opc2<1:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|------------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b1:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_AP0R[UInt(opc2<1:0>)] = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_AP0R[UInt(opc2<1:0>)] = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_AP0R[UInt(opc2<1:0>)] = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_AP0R[UInt(opc2<1:0>)] = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            ICC_AP0R[UInt(opc2<1:0>)] = R[t];

```

ICC_AP1R<n>, Interrupt Controller Active Priorities Group 1 Registers, n = 0 - 3

The ICC_AP1R<n> characteristics are:

Purpose

Provides information about Group 1 active priorities.

Configuration

AArch32 System register ICC_AP1R<n> bits [31:0] (S) are architecturally mapped to AArch64 System register [ICC_AP1R<n>_EL1\[31:0\]](#) (S).

AArch32 System register ICC_AP1R<n> bits [31:0] (NS) are architecturally mapped to AArch64 System register [ICC_AP1R<n>_EL1\[31:0\]](#) (NS).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_AP1R<n> are UNDEFINED.

Attributes

ICC_AP1R<n> is a 32-bit register.

Field descriptions

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

Accessing ICC_AP1R<n>

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 1 active priorities) might result in UNPREDICTABLE behavior of the interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICC_AP1R1 is only implemented in implementations that support 6 or more bits of preemption. ICC_AP1R2 and ICC_AP1R3 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by [ICH_VTR](#).PREbits.

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- [ICC_AP0R<n>](#)
- Secure ICC_AP1R<n>
- Non-secure ICC_AP1R<n>

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1100 | 0b1001 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_AP1R[UInt(opc2<1:0>)];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_AP1R[UInt(opc2<1:0>)];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            return ICC_AP1R_NS[UInt(opc2<1:0>)];
        else
            return ICC_AP1R[UInt(opc2<1:0>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            return ICC_AP1R_NS[UInt(opc2<1:0>)];
        else
            return ICC_AP1R[UInt(opc2<1:0>)];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            if SCR.NS == '0' then
                return ICC_AP1R_S[UInt(opc2<1:0>)];
            else
                return ICC_AP1R_NS[UInt(opc2<1:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1100 | 0b1001 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_AP1R[UInt(opc2<1:0>)] = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_AP1R[UInt(opc2<1:0>)] = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            ICC_AP1R_NS[UInt(opc2<1:0>)] = R[t];
        else
            ICC_AP1R[UInt(opc2<1:0>)] = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            ICC_AP1R_NS[UInt(opc2<1:0>)] = R[t];
        else
            ICC_AP1R[UInt(opc2<1:0>)] = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            if SCR.NS == '0' then
                ICC_AP1R_S[UInt(opc2<1:0>)] = R[t];
            else
                ICC_AP1R_NS[UInt(opc2<1:0>)] = R[t];

```


ICC_ASGI1R, Interrupt Controller Alias Software Generated Interrupt Group 1 Register

The ICC_ASGI1R characteristics are:

Purpose

Generates Group 1 SGIs for the Security state that is not the current Security state.

Configuration

AArch32 System register ICC_ASGI1R performs the same function as AArch64 System register [ICC_ASGI1R_EL1](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_ASGI1R are UNDEFINED.

Under certain conditions a write to ICC_ASGI1R can generate Group 0 interrupts, see 'Forwarding an SGI to a target PE' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_ASGI1R is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|----|----|------|----|----|----|----|----|----|----|------------|----|----|----|------|----|-----|------|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | Aff3 | | | | | | | | RS | | | | RES0 | | IRM | Aff2 | | | | | | | | |
| RES0 | | | | INTID | | | | Aff1 | | | | | | | | TargetList | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:56]

Reserved, RES0.

Aff3, bits [55:48]

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

RS, bits [47:44]

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value $((RS * 16) + n)$.

When [ICC_CTLR_EL1](#).RSS==0, RS is RES0.

When [ICC_CTLR_EL1](#).RSS==1 and [GICD_TYPER](#).RSS==0, writing this register with $RS \neq 0$ is a CONSTRAINED UNPREDICTABLE choice of :

- The write is ignored.
- The RS field is treated as 0.

Bits [43:41]

Reserved, RES0.

IRM, bit [40]

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

| IRM | Meaning |
|-----|---|
| 0b0 | Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.<target list>. |
| 0b1 | Interrupts routed to all PEs in the system, excluding "self". |

Aff2, bits [39:32]

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]

Reserved, RES0.

INTID, bits [27:24]

The INTID of the SGI.

Aff1, bits [23:16]

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Note

If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

Accessing ICC_ASGI1R

This register allows software executing in a Secure state to generate Non-secure Group 1 SGIs. It will also allow software executing in a Non-secure state to generate Secure Group 1 SGIs, if permitted by the settings of [GICR_NSACR](#) in the Redistributor corresponding to the target PE.

When [GICD_CTLR.DS](#)=0, Non-secure writes do not generate an interrupt for a target PE if not permitted by the [GICR_NSACR](#) register associated with the target PE. For more information, see 'Use of control registers for SGI forwarding' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Note

Accesses from Secure Monitor mode are treated as Secure regardless of the value of SCR.NS.

Accesses to this register use the following encodings in the System register encoding space:

MCCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|---------------|------------|-------------|
| 0b1111 | 0b1100 | 0b0001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_ASGI1R = R[t2]:R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_ASGI1R = R[t2]:R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            ICC_ASGI1R = R[t2]:R[t];

```


ICC_BPR0, Interrupt Controller Binary Point Register 0

The ICC_BPR0 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption.

Configuration

AArch32 System register ICC_BPR0 bits [31:0] are architecturally mapped to AArch64 System register [ICC_BPR0_EL1\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_BPR0 are UNDEFINED.

Attributes

ICC_BPR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | BinaryPoint | | | | | | | | | | | | | | |

Bits [31:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. This is done as follows:

| Binary point value | Group priority field | Subpriority field | Field with binary point |
|--------------------|----------------------|-------------------|-------------------------|
| 0 | [7:1] | [0] | ggggggg.s |
| 1 | [7:2] | [1:0] | gggggg.ss |
| 2 | [7:3] | [2:0] | ggggg.sss |
| 3 | [7:4] | [3:0] | gggg.ssss |
| 4 | [7:5] | [4:0] | ggg.sssss |
| 5 | [7:6] | [5:0] | gg.ssssss |
| 6 | [7] | [6:0] | g.sssssss |
| 7 | No preemption | [7:0] | .ssssssss |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICC_BPR0

The minimum binary point value is derived from the number of implemented priority bits. The number of priority bits is IMPLEMENTATION DEFINED, and reported by [ICC_CTLR.PRIBits](#) and [ICC_MCTLR.PRIBits](#).

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is set to the minimum supported value.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_BPR0;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_BPR0;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_BPR0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_BPR0;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_BPR0;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_BPR0 = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_BPR0 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_BPR0 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_BPR0 = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            ICC_BPR0 = R[t];

```


| HCR.IMO | SCR_IRQ | Behavior |
|---------|---------|--|
| 0b0 | 0b0 | Non-secure EL1 and EL2 reads return ICC_BPR0 + 1 saturated to 0b111. Non-secure EL1 and EL2 writes are ignored. |
| 0b0 | 0b1 | Non-secure EL1 and EL2 accesses trap to EL3. |
| 0b1 | 0b0 | Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 reads return ICC_BPR0 + 1 saturated to 0b111. Non-secure EL2 writes are ignored. |
| 0b1 | 0b1 | Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 accesses trap to EL3. |

If EL3 is not implemented and [ICC_CTLR](#).CBPR is 1, Non-secure accesses to this register at EL1 or EL2 behave as follows, depending on the values of HCR.IMO:

| HCR.IMO | Behavior |
|---------|--|
| 0b0 | Non-secure EL1 and EL2 reads return ICC_BPR0 + 1 saturated to 0b111. Non-secure EL1 and EL2 writes are ignored. |
| 0b1 | Non-secure EL1 accesses affect virtual interrupts. Non-secure EL2 reads return ICC_BPR0 + 1 saturated to 0b111. Non-secure EL2 writes are ignored. |

This field resets to an IMPLEMENTATION DEFINED non-zero value.

Accessing ICC_BPR1

When the PE resets into an Exception level that is using AArch32, the reset value is equal to:

- For the Secure copy of the register, the minimum value of [ICC_BPR0](#) plus one.
- For the Non-secure copy of the register, the minimum value of [ICC_BPR0](#).

Where the minimum value of [ICC_BPR0](#) is IMPLEMENTATION DEFINED.

If EL3 is not implemented:

- If the PE is Secure this reset value is (minimum value of [ICC_BPR0](#) plus one).
- If the PE is Non-secure this reset value is (minimum value of [ICC_BPR0](#)).

An attempt to program the binary point field to a value less than the reset value sets the field to the reset value.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_BPR1;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_BPR1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        return ICC_BPR1_NS;
    else
        return ICC_BPR1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        return ICC_BPR1_NS;
    else
        return ICC_BPR1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            return ICC_BPR1_S;
        else
            return ICC_BPR1_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_BPR1 = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_BPR1 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            ICC_BPR1_NS = R[t];
        else
            ICC_BPR1 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            ICC_BPR1_NS = R[t];
        else
            ICC_BPR1 = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            if SCR.NS == '0' then
                ICC_BPR1_S = R[t];
            else
                ICC_BPR1_NS = R[t];

```


ICC_CTLR, Interrupt Controller Control Register

The ICC_CTLR characteristics are:

Purpose

Controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

Configuration

AArch32 System register ICC_CTLR bits [31:0] (S) are architecturally mapped to AArch64 System register [ICC_CTLR_EL1\[31:0\]](#) (S).

AArch32 System register ICC_CTLR bits [31:0] (NS) are architecturally mapped to AArch64 System register [ICC_CTLR_EL1\[31:0\]](#) (NS).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_CTLR are UNDEFINED.

Attributes

ICC_CTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----------|-----|------|-----|------|--------|---------|------|------|------|---------|------|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | ExtRange | RSS | RES0 | A3V | SEIS | IDbits | PRIbits | RES0 | PMHE | RES0 | EOImode | CBPR | | | | | | | | |

Bits [31:20]

Reserved, RES0.

ExtRange, bit [19]

Extended INTID range (read-only).

| ExtRange | Meaning |
|----------|--|
| 0b0 | CPU interface does not support INTIDs in the range 1024..8191. Behaviour is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface. Note Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them. |
| 0b1 | CPU interface supports INTIDs in the range 1024..8191. All INTIDs in the range 1024..8191 are treated as requiring deactivation. |

If EL3 is implemented, ICC_CTLR_EL1.ExtRange is an alias of [ICC_CTLR_EL3.ExtRange](#).

RSS, bit [18]

Range Selector Support. Possible values are:

| RSS | Meaning |
|-----|--|
| 0b0 | Targeted SGIs with affinity level 0 values of 0 - 15 are supported. |
| 0b1 | Targeted SGIs with affinity level 0 values of 0 - 255 are supported. |

This bit is read-only.

Bits [17:16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored. Possible values are:

| A3V | Meaning |
|-----|---|
| 0b0 | The CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers. |
| 0b1 | The CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers. |

If EL3 is implemented and using AArch32, this bit is an alias of [ICC_MCTLR.A3V](#).

If EL3 is implemented and using AArch64, this bit is an alias of [ICC_CTLR_EL3.A3V](#).

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the CPU interface supports local generation of SEIs:

| SEIS | Meaning |
|------|--|
| 0b0 | The CPU interface logic does not support local generation of SEIs. |
| 0b1 | The CPU interface logic supports local generation of SEIs. |

If EL3 is implemented and using AArch32, this bit is an alias of [ICC_MCTLR.SEIS](#).

If EL3 is implemented and using AArch64, this bit is an alias of [ICC_CTLR_EL3.SEIS](#).

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. The number of physical interrupt identifier bits supported:

| IDbits | Meaning |
|--------|----------|
| 0b000 | 16 bits. |
| 0b001 | 24 bits. |

All other values are reserved.

If EL3 is implemented and using AArch32, this field is an alias of [ICC_MCTLR.IDbits](#).

If EL3 is implemented and using AArch64, this field is an alias of [ICC_CTLR_EL3.IDbits](#).

PRibits, bits [10:8]

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation that supports two Security states must implement at least 32 levels of physical priority (5 priority bits).

An implementation that supports only a single Security state must implement at least 16 levels of physical priority (4 priority bits).

Note

This field always returns the number of priority bits implemented, regardless of the Security state of the access or the value of [GICD_CTLR.DS](#).

The division between group priority and subpriority is defined in the binary point registers [ICC_BPR0](#) and [ICC_BPR1](#).

If EL3 is implemented and using AArch32, physical accesses return the value from [ICC_MCTLR.PRIBits](#).

If EL3 is implemented and using AArch64, physical accesses return the value from [ICC_CTLR_EL3.PRIBits](#).

If EL3 is not implemented, physical accesses return the value from this field.

Bit [7]

Reserved, RES0.

PMHE, bit [6]

Priority Mask Hint Enable. Controls whether the priority mask register is used as a hint for interrupt distribution:

| PMHE | Meaning |
|------|---|
| 0b0 | Disables use of ICC_PMR as a hint for interrupt distribution. |
| 0b1 | Enables use of ICC_PMR as a hint for interrupt distribution. |

If EL3 is implemented:

- If EL3 is using AArch32, this bit is an alias of [ICC_MCTLR.PMHE](#).
- If EL3 is using AArch64, this bit is an alias of [ICC_CTLR_EL3.PMHE](#).
- If [GICD_CTLR.DS](#) == 0, this bit is read-only.
- If [GICD_CTLR.DS](#) == 1, this bit is read/write.

If EL3 is not implemented, it is IMPLEMENTATION DEFINED whether this bit is read-only or read-write:

- If this bit is read-only, an implementation can choose to make this field RAZ/WI or RAO/WI.
- If this bit is read/write, it resets to zero.

Bits [5:2]

Reserved, RES0.

EOImode, bit [1]

EOI mode for the current Security state. Controls whether a write to an End of Interrupt register also deactivates the interrupt:

| EOImode | Meaning |
|---------|---|
| 0b0 | ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are UNPREDICTABLE. |
| 0b1 | ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides interrupt deactivation functionality. |

If EL3 is implemented:

- If EL3 is using AArch32, this bit is an alias of [ICC_MCTLR.EOImode_EL1](#){S, NS} where S or NS corresponds to the current Security state.
- If EL3 is using AArch64, this bit is an alias of [ICC_CTLR_EL3.EOImode_EL1](#){S, NS} where S or NS corresponds to the current Security state.

If EL3 is not implemented, it is IMPLEMENTATION DEFINED whether this bit is read-only or read-write:

- If this bit is read-only, an implementation can choose to make this field RAZ/WI or RAO/WI.
- If this bit is read/write, it resets to zero.

CBPR, bit [0]

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 interrupts:

| CBPR | Meaning |
|------|---|
| 0b0 | ICC_BPR0 determines the preemption group for Group 0 interrupts only. ICC_BPR1 determines the preemption group for Group 1 interrupts. |
| 0b1 | ICC_BPR0 determines the preemption group for both Group 0 and Group 1 interrupts. |

If EL3 is implemented:

- If EL3 is using AArch32, this bit is an alias of [ICC_MCTLR.CBPR_EL1](#){S,NS} where S or NS corresponds to the current Security state.
- If EL3 is using AArch64, this bit is an alias of [ICC_CTLR_EL3.CBPR_EL1](#){S,NS} where S or NS corresponds to the current Security state.
- If [GICD_CTLR.DS](#) == 0, this bit is read-only.
- If [GICD_CTLR.DS](#) == 1, this bit is read/write.

If EL3 is not implemented, it is IMPLEMENTATION DEFINED whether this bit is read-only or read-write:

- If this bit is read-only, an implementation can choose to make this field RAZ/WI or RAO/WI.
- If this bit is read/write, it resets to zero.

Accessing ICC_CTLR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_CTLR;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_CTLR;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_CTLR;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_CTLR;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        return ICC_CTLR_NS;
    else
        return ICC_CTLR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        return ICC_CTLR_NS;
    else
        return ICC_CTLR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            return ICC_CTLR_S;

```

```
else
    return ICC_CTLR_NS;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_CTLR = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_CTLR = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_CTLR = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_CTLR = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_CTLR_NS = R[t];
    else
        ICC_CTLR = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_CTLR_NS = R[t];
    else
        ICC_CTLR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            ICC_CTLR_S = R[t];

```

```
else  
    ICC_CTLR_NS = R[t];
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_DIR, Interrupt Controller Deactivate Interrupt Register

The ICC_DIR characteristics are:

Purpose

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified interrupt.

Configuration

AArch32 System register ICC_DIR performs the same function as AArch64 System register [ICC_DIR_EL1](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_DIR are UNDEFINED.

Attributes

ICC_DIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the interrupt to be deactivated.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR.IDbits](#) and [ICC_MCTLR.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_DIR

There are two cases when writing to [ICC_DIR_EL1](#) that were UNPREDICTABLE for a corresponding GICv2 write to [GICC_DIR](#):

- When EOImode == 0. GICv3 implementations must ignore such writes. In systems supporting system error generation, an implementation might generate an SEI.
- When EOImode == 1 but no EOI has been issued. The interrupt will be de-activated by the Distributor, however the active priority in the CPU interface for the interrupt will remain set (because no EOI was issued).

Accesses to this register use the following encodings in the System register encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TDIR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TDIR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_DIR = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_DIR = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_DIR = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_DIR = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
            else
                ICC_DIR = R[t];
        elsif PSTATE.EL == EL2 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
                UNDEFINED;
            elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
                UNDEFINED;
            elsif ICC_HSRE.SRE == '0' then
                UNDEFINED;
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch32.TakeMonitorTrapException();
            else
                ICC_DIR = R[t];
        elsif PSTATE.EL == EL3 then
            if ICC_MSRE.SRE == '0' then
                UNDEFINED;
            else
                ICC_DIR = R[t];

```


ICC_EOIR0, Interrupt Controller End Of Interrupt Register 0

The ICC_EOIR0 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified Group 0 interrupt.

Configuration

AArch32 System register ICC_EOIR0 performs the same function as AArch64 System register [ICC_EOIR0_EL1](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_EOIR0 are UNDEFINED.

Attributes

ICC_EOIR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding [ICC_IAR0](#) access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR.IDbits](#) and [ICC_MCTLR.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the EOImode bit for the current Exception level and Security state is 0, a write to this register drops the priority for the interrupt, and also deactivates the interrupt.

If the EOImode bit for the current Exception level and Security state is 1, a write to this register only drops the priority for the interrupt. Software must write to [ICC_DIR](#) to deactivate the interrupt.

The appropriate EOImode bit varies as follows:

- If EL3 is not implemented, the appropriate bit is [ICC_CTLR.EOImode](#).
- If EL3 is implemented and the software is executing in Monitor mode, the appropriate bit is [ICC_MCTLR.EOImode_EL3](#).
- If EL3 is implemented and the software is not executing in Monitor mode, the bit depends on the current Security state:
 - If the software is executing in Secure state, the bit is [ICC_CTLR.EOImode](#) in the Secure instance of [ICC_CTLR](#). This is an alias of [ICC_MCTLR.EOImode_EL1S](#).
 - If the software is executing in Non-secure state, the bit is [ICC_CTLR.EOImode](#) in the Non-secure instance of [ICC_CTLR](#). This is an alias of [ICC_MCTLR.EOImode_EL1NS](#).

Accessing ICC_EOIR0

A write to this register must correspond to the most recent valid read by this PE from an Interrupt Acknowledge Register, and must correspond to the INTID that was read from [ICC_IAR0](#), otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

A write of a Special INTID is ignored. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Accesses to this register use the following encodings in the System register encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICC_EOIR0 = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICC_EOIR0 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_EOIR0 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_EOIR0 = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            ICC_EOIR0 = R[t];

```

ICC_EOIR1, Interrupt Controller End Of Interrupt Register 1

The ICC_EOIR1 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified Group 1 interrupt.

Configuration

AArch32 System register ICC_EOIR1 performs the same function as AArch64 System register [ICC_EOIR1_EL1](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_EOIR1 are UNDEFINED.

Attributes

ICC_EOIR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding [ICC_IAR1](#) access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR.IDbits](#) and [ICC_MCTLR.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the EOImode bit for the current Exception level and Security state is 0, a write to this register drops the priority for the interrupt, and also deactivates the interrupt.

If the EOImode bit for the current Exception level and Security state is 1, a write to this register only drops the priority for the interrupt. Software must write to [ICC_DIR](#) to deactivate the interrupt.

The appropriate EOImode bit varies as follows:

- If EL3 is not implemented, the appropriate bit is [ICC_CTLR.EOImode](#).
- If EL3 is implemented and the software is executing in Monitor mode, the appropriate bit is [ICC_MCTLR.EOImode_EL3](#).
- If EL3 is implemented and the software is not executing in Monitor mode, the bit depends on the current Security state:
 - If the software is executing in Secure state, the bit is [ICC_CTLR.EOImode](#) in the Secure instance of [ICC_CTLR](#). This is an alias of [ICC_MCTLR.EOImode_EL1S](#).
 - If the software is executing in Non-secure state, the bit is [ICC_CTLR.EOImode](#) in the Non-secure instance of [ICC_CTLR](#). This is an alias of [ICC_MCTLR.EOImode_EL1NS](#).

Accessing ICC_EOIR1

A write to this register must correspond to the most recent valid read by this PE from an Interrupt Acknowledge Register, and must correspond to the INTID that was read from [ICC_IAR1](#), otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

A write of a Special INTID is ignored. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Accesses to this register use the following encodings in the System register encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_EOIR1 = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_EOIR1 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_EOIR1 = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_EOIR1 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_EOIR1 = R[t];

```

ICC_HPPIR0, Interrupt Controller Highest Priority Pending Interrupt Register 0

The ICC_HPPIR0 characteristics are:

Purpose

Indicates the highest priority pending Group 0 interrupt on the CPU interface.

Configuration

AArch32 System register ICC_HPPIR0 performs the same function as AArch64 System register [ICC_HPPIR0_EL1](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_HPPIR0 are UNDEFINED.

Attributes

ICC_HPPIR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending interrupt, if that interrupt is observable at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR.IDbits](#) and [ICC_MCTLR.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_HPPIR0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_HPPIR0;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_HPPIR0;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_HPPIR0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_HPPIR0;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_HPPIR0;

```

ICC_HPPIR1, Interrupt Controller Highest Priority Pending Interrupt Register 1

The ICC_HPPIR1 characteristics are:

Purpose

Indicates the highest priority pending Group 1 interrupt on the CPU interface.

Configuration

AArch32 System register ICC_HPPIR1 performs the same function as AArch64 System register [ICC_HPPIR1_EL1](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_HPPIR1 are UNDEFINED.

Attributes

ICC_HPPIR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending interrupt, if that interrupt is observable at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR.IDbits](#) and [ICC_MCTLR.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_HPPIR1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_HPPIR1;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_HPPIR1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_HPPIR1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_HPPIR1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_HPPIR1;

```

ICC_HSRE, Interrupt Controller Hyp System Register Enable register

The ICC_HSRE characteristics are:

Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL2.

Configuration

AArch32 System register ICC_HSRE bits [31:0] are architecturally mapped to AArch64 System register [ICC_SRE_EL2\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICC_HSRE are UNDEFINED.

Attributes

ICC_HSRE is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|-----|-----|-----|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | Enable | | DIB | DFB | SRE | | | | | | | | | |

Bits [31:4]

Reserved, RES0.

Enable, bit [3]

Enable. Enables lower Exception level access to [ICC_SRE](#).

| Enable | Meaning |
|--------|--|
| 0b0 | Non-secure EL1 accesses to ICC_SRE trap to EL2. |
| 0b1 | Non-secure EL1 accesses to ICC_SRE do not trap to EL2. |

If ICC_HSRE.SRE is RAO/WI, an implementation is permitted to make the Enable bit RAO/WI.

If ICC_HSRE.SRE is 0, the Enable bit behaves as 1 for all purposes other than reading the value of the bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIB, bit [2]

Disable IRQ bypass.

| DIB | Meaning |
|-----|----------------------|
| 0b0 | IRQ bypass enabled. |
| 0b1 | IRQ bypass disabled. |

If EL3 is implemented and [GICD_CTLR](#).DS is 0, this field is a read-only alias of [ICC_MSRE](#).DIB.

If EL3 is implemented and [GICD_CTLR.DS](#) is 1, this field is a read-write alias of [ICC_MSRE.DIB](#).

In systems that do not support IRQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

DFB, bit [1]

Disable FIQ bypass.

| DFB | Meaning |
|-----|----------------------|
| 0b0 | FIQ bypass enabled. |
| 0b1 | FIQ bypass disabled. |

If EL3 is implemented and [GICD_CTLR.DS](#) is 0, this field is a read-only alias of [ICC_MSRE.DFB](#).

If EL3 is implemented and [GICD_CTLR.DS](#) is 1, this field is a read-write alias of [ICC_MSRE.DFB](#).

In systems that do not support FIQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

SRE, bit [0]

System Register Enable.

| SRE | Meaning |
|-----|--|
| 0b0 | The memory-mapped interface must be used. Accesses at EL2 or below to any ICH_* System register, or any EL1 or EL2 ICC_* register other than ICC_SRE or ICC_HSRE, are UNDEFINED. |
| 0b1 | The System register interface to the ICH_* registers and the EL1 and EL2 ICC_* registers is enabled for EL2. |

If software changes this bit from 1 to 0, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

If EL3 is implemented and using AArch64:

- When [ICC_SRE_EL3.SRE](#)==0 this bit is RAZ/WI.

If EL3 is implemented using AArch32:

- When [ICC_MSRE.SRE](#)==0 this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_HSRE

The GIC architecture permits, but does not require, that registers can be shared between memory-mapped registers and the equivalent System registers. This means that if the memory-mapped registers have been accessed while [ICC_HSRE.SRE](#)==0, then the System registers might be modified. Therefore, software must only rely on the reset values of the System registers if there has been no use of the GIC functionality while the memory-mapped registers are in use. Otherwise, the System register values must be treated as UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b1001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif ICC_MSRE.Enable == '0' then
        UNDEFINED;
    else
        return ICC_HSRE;
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        return ICC_HSRE;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b1001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif ICC_MSRE.Enable == '0' then
        UNDEFINED;
    else
        ICC_HSRE = R[t];
elsif PSTATE.EL == EL3 then
    if !EL2Enabled() then
        UNDEFINED;
    else
        ICC_HSRE = R[t];

```


ICC_IAR0, Interrupt Controller Interrupt Acknowledge Register 0

The ICC_IAR0 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled Group 0 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch32 System register ICC_IAR0 performs the same function as AArch64 System register [ICC_IAR0_EL1](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_IAR0 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when $PSTATE.\{I,F\} == \{0,0\}$). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_IAR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

This is the INTID of the highest priority pending interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. These special INTIDs can be one of: 1020, 1021, or 1023. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR.IDbits](#) and [ICC_MCTLR.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_IAR0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICC_IAR0;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICC_IAR0;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_IAR0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_IAR0;
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            return ICC_IAR0;

```


ICC_IAR1, Interrupt Controller Interrupt Acknowledge Register 1

The ICC_IAR1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled Group 1 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch32 System register ICC_IAR1 performs the same function as AArch64 System register [ICC_IAR1_EL1](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_IAR1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when $PSTATE.\{I,F\} == \{0,0\}$). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_IAR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

This is the INTID of the highest priority pending interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged at the current Security state and Exception level.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICC_CTLR.IDbits](#) and [ICC_MCTLR.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICC_IAR1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_IAR1;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_IAR1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_IAR1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_IAR1;
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            return ICC_IAR1;

```


ICC_IGRPEN0, Interrupt Controller Interrupt Group 0 Enable register

The ICC_IGRPEN0 characteristics are:

Purpose

Controls whether Group 0 interrupts are enabled or not.

Configuration

AArch32 System register ICC_IGRPEN0 bits [31:0] are architecturally mapped to AArch64 System register [ICC_IGRPEN0_EL1\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_IGRPEN0 are UNDEFINED.

Attributes

ICC_IGRPEN0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | Enable | | | | | | | | | | | | | | | |

Bits [31:1]

Reserved, RES0.

Enable, bit [0]

Enables Group 0 interrupts.

| Enable | Meaning |
|--------|----------------------------------|
| 0b0 | Group 0 interrupts are disabled. |
| 0b1 | Group 0 interrupts are enabled. |

Virtual accesses to this register update [ICH_VMCR.VENG0](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_IGRPEN0

The lowest Exception level at which this register can be accessed is governed by the Exception level to which FIQ is routed. This routing depends on SCR.FIQ, SCR.NS and HCR.FMO.

If an interrupt is pending within the CPU interface when Enable becomes 0, the interrupt must be released to allow the Distributor to forward the interrupt to a different PE.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICC_IGRPEN0;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICC_IGRPEN0;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_IGRPEN0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_IGRPEN0;
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            return ICC_IGRPEN0;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_IGRPEN0 = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_IGRPEN0 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_IGRPEN0 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_IGRPEN0 = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            ICC_IGRPEN0 = R[t];

```


ICC_IGRPEN1, Interrupt Controller Interrupt Group 1 Enable register

The ICC_IGRPEN1 characteristics are:

Purpose

Controls whether Group 1 interrupts are enabled for the current Security state.

Configuration

AArch32 System register ICC_IGRPEN1 bits [31:0] (S) are architecturally mapped to AArch64 System register [ICC_IGRPEN1_EL1\[31:0\]](#) (S).

AArch32 System register ICC_IGRPEN1 bits [31:0] (NS) are architecturally mapped to AArch64 System register [ICC_IGRPEN1_EL1\[31:0\]](#) (NS).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_IGRPEN1 are UNDEFINED.

Attributes

ICC_IGRPEN1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|--------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Enable |

Bits [31:1]

Reserved, RES0.

Enable, bit [0]

Enables Group 1 interrupts for the current Security state.

| Enable | Meaning |
|--------|---|
| 0b0 | Group 1 interrupts are disabled for the current Security state. |
| 0b1 | Group 1 interrupts are enabled for the current Security state. |

Virtual accesses to this register update [ICH_VMCR.VENG1](#).

If EL3 is present:

- This bit is a read/write alias of [ICC_MGRPEN1](#).EnableGrp1{S, NS} as appropriate if EL3 is using AArch32, or [ICC_IGRPEN1_EL3](#).EnableGrp1{S, NS} as appropriate if EL3 is using AArch64.
- When this register is accessed at EL3, the copy of this register appropriate to the current setting of SCR.NS is accessed.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_IGRPEN1

The lowest Exception level at which this register can be accessed is governed by the Exception level to which IRQ is routed. This routing depends on SCR.IRQ, SCR.NS and HCR.IMO.

If an interrupt is pending within the CPU interface when Enable becomes 0, the interrupt must be released to allow the Distributor to forward the interrupt to a different PE.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_IGRPEN1;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_IGRPEN1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            return ICC_IGRPEN1_NS;
        else
            return ICC_IGRPEN1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            return ICC_IGRPEN1_NS;
        else
            return ICC_IGRPEN1;
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            if SCR.NS == '0' then
                return ICC_IGRPEN1_S;
            else
                return ICC_IGRPEN1_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_IGRPEN1 = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_IGRPEN1 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_IGRPEN1_NS = R[t];
    else
        ICC_IGRPEN1 = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        ICC_IGRPEN1_NS = R[t];
    else
        ICC_IGRPEN1 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            ICC_IGRPEN1_S = R[t];
        else
            ICC_IGRPEN1_NS = R[t];

```


ICC_MCTLR, Interrupt Controller Monitor Control Register

The ICC_MCTLR characteristics are:

Purpose

Controls aspects of the behavior of the GIC CPU interface and provides information about the features implemented.

Configuration

AArch32 System register ICC_MCTLR bits [31:0] can be mapped to AArch64 System register [ICC_CTLR_EL3\[31:0\]](#), but this is not architecturally mandated.

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL3 is implemented. Otherwise, direct accesses to ICC_MCTLR are UNDEFINED.

Attributes

ICC_MCTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----------|-----|-----|------|-----|------|--------|---------|------|------|----|-------------|----|-------------|----|-------------|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 |
| RES0 | | | | | | | | | | | | ExtRange | RSS | nDS | RES0 | A3V | SEIS | IDbits | PRIbits | RES0 | PMHE | RM | EOImode_EL1 | NS | EOImode_EL1 | NS | EOImode_EL1 | NS |

Bits [31:20]

Reserved, RES0.

ExtRange, bit [19]

Extended INTID range (read-only).

| ExtRange | Meaning |
|----------|--|
| 0b0 | <p>CPU interface does not support INTIDs in the range 1024..8191. Behavior is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface.</p> <p>Note Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.</p> |
| 0b1 | <p>CPU interface supports INTIDs in the range 1024..8191 All INTIDs in the range 1024..8191 are treated as requiring deactivation.</p> |

RSS, bit [18]

Range Selector Support. Possible values are:

| RSS | Meaning |
|------------|--|
| 0b0 | Targeted SGIs with affinity level 0 values of 0 - 15 are supported. |
| 0b1 | Targeted SGIs with affinity level 0 values of 0 - 255 are supported. |

This bit is read-only.

nDS, bit [17]

Disable Security not supported. Read-only and writes are ignored.

| nDS | Meaning |
|------------|---|
| 0b0 | The CPU interface logic supports disabling of security. |
| 0b1 | The CPU interface logic does not support disabling of security, and requires that security is not disabled. |

Bit [16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored.

| A3V | Meaning |
|------------|--|
| 0b0 | The CPU interface logic does not support non-zero values of the Aff3 field in SGI generation System registers. |
| 0b1 | The CPU interface logic supports non-zero values of the Aff3 field in SGI generation System registers. |

If EL3 is present, [ICC_CTLR.A3V](#) is an alias of ICC_MCTLR.A3V

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the CPU interface supports generation of SEIs.

| SEIS | Meaning |
|-------------|--|
| 0b0 | The CPU interface logic does not support generation of SEIs. |
| 0b1 | The CPU interface logic supports generation of SEIs. |

If EL3 is present, [ICC_CTLR.SEIS](#) is an alias of ICC_MCTLR.SEIS

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. Indicates the number of physical interrupt identifier bits supported.

| IDbits | Meaning |
|---------------|----------------|
| 0b000 | 16 bits. |
| 0b001 | 24 bits. |

All other values are reserved.

If EL3 is present, [ICC_CTLR.IDbits](#) is an alias of ICC_MCTLR.IDbits

PRibits, bits [10:8]

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation that supports two Security states must implement at least 32 levels of physical priority (5 priority bits).

An implementation that supports only a single Security state must implement at least 16 levels of physical priority (4 priority bits).

Note

This field always returns the number of priority bits implemented, regardless of the value of SCR.NS or the value of [GICD_CTLR.DS](#).

The division between group priority and subpriority is defined in the binary point registers [ICC_BPR0](#) and [ICC_BPR1](#).

This field determines the minimum value of ICC_BPR0.

Bit [7]

Reserved, RES0.

PMHE, bit [6]

Priority Mask Hint Enable.

| PMHE | Meaning |
|------|--|
| 0b0 | Disables use of the priority mask register as a hint for interrupt distribution. |
| 0b1 | Enables use of the priority mask register as a hint for interrupt distribution. |

Software must write [ICC_PMR](#) to 0xFF before clearing this field to 0.

An implementation might choose to make this field RAO/WI.

If EL3 is present, [ICC_CTLR](#).PMHE is an alias of ICC_MCTLR.PMHE.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

RM, bit [5]

SBZ.

The equivalent bit in AArch64 is the Routing Modifier bit. This feature is not supported when EL3 is using AArch32.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EOImode_EL1NS, bit [4]

EOI mode for interrupts handled at Non-secure EL1 and EL2. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

| EOImode_EL1NS | Meaning |
|---------------|---|
| 0b0 | ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are UNPREDICTABLE. |
| 0b1 | ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides interrupt deactivation functionality. |

If EL3 is present, [ICC_CTLR\(NS\)](#).EOImode is an alias of ICC_MCTLR.EOImode_EL1NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EOImode_EL1S, bit [3]

EOI mode for interrupts handled at Secure EL1. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

| EOImode_EL1S | Meaning |
|--------------|---|
| 0b0 | ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are UNPREDICTABLE. |
| 0b1 | ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides interrupt deactivation functionality. |

If EL3 is present, [ICC_CTLR\(S\)](#).EOImode is an alias of ICC_MCTLR.EOImode_EL1S.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EOImode_EL3, bit [2]

EOI mode for interrupts handled at EL3. Controls whether a write to an End of Interrupt register also deactivates the interrupt.

| EOImode_EL3 | Meaning |
|-------------|---|
| 0b0 | ICC_EOIR0 and ICC_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICC_DIR are UNPREDICTABLE. |
| 0b1 | ICC_EOIR0 and ICC_EOIR1 provide priority drop functionality only. ICC_DIR provides interrupt deactivation functionality. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CBPR_EL1NS, bit [1]

Common Binary Point Register, EL1 Non-secure. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 Non-secure interrupts at EL1 and EL2.

| CBPR_EL1NS | Meaning |
|------------|--|
| 0b0 | ICC_BPR0 determines the preemption group for Group 0 interrupts only. ICC_BPR1 determines the preemption group for Non-secure Group 1 interrupts. |
| 0b1 | ICC_BPR0 determines the preemption group for Group 0 interrupts and Non-secure Group 1 interrupts. Non-secure accesses to GICC_BPR and ICC_BPR1 access the state of ICC_BPR0 . |

If EL3 is present, [ICC_CTLR\(NS\)](#).CBPR is an alias of ICC_MCTLR.CBPR_EL1NS.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CBPR_EL1S, bit [0]

Common Binary Point Register, EL1 Secure. Controls whether the same register is used for interrupt preemption of both Group 0 and Group 1 Secure interrupts in Secure non-Monitor modes.

| CBPR_EL1S | Meaning |
|-----------|--|
| 0b0 | ICC_BPR0 determines the preemption group for Group 0 interrupts only. ICC_BPR1 determines the preemption group for Secure Group 1 interrupts. |
| 0b1 | ICC_BPR0 determines the preemption group for Group 0 interrupts and Secure Group 1 interrupts. Secure EL1 accesses, or EL3 accesses when not in Monitor mode, to ICC_BPR1 access the state of ICC_BPR0 . |

If EL3 is present, [ICC_CTLR\(S\)](#).CBPR is an alias of ICC_MCTLR.CBPR_EL1S.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICC_MCTLR

This register is only accessible when executing in Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b110 | 0b1100 | 0b1100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_MCTLR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b110 | 0b1100 | 0b1100 | 0b100 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_MCTLR = R[t];
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_MGRPEN1, Interrupt Controller Monitor Interrupt Group 1 Enable register

The ICC_MGRPEN1 characteristics are:

Purpose

Controls whether Group 1 interrupts are enabled or not.

Configuration

AArch32 System register ICC_MGRPEN1 bits [31:0] can be mapped to AArch64 System register [ICC_IGRPEN1_EL3\[31:0\]](#), but this is not architecturally mandated.

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL3 is implemented. Otherwise, direct accesses to ICC_MGRPEN1 are UNDEFINED.

Attributes

ICC_MGRPEN1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|--------------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | EnableGrp1S | | EnableGrp1NS | | | | | | | | | | | | | |

Bits [31:2]

Reserved, RES0.

EnableGrp1S, bit [1]

Enables Group 1 interrupts for the Secure state.

| EnableGrp1S | Meaning |
|-------------|---|
| 0b0 | Secure Group 1 interrupts are disabled. |
| 0b1 | Secure Group 1 interrupts are enabled. |

The Secure [ICC_IGRPEN1](#).Enable bit is a read/write alias of the ICC_MGRPEN1.EnableGrp1S bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

EnableGrp1NS, bit [0]

Enables Group 1 interrupts for the Non-secure state.

| EnableGrp1NS | Meaning |
|--------------|---|
| 0b0 | Non-secure Group 1 interrupts are disabled. |
| 0b1 | Non-secure Group 1 interrupts are enabled. |

The Non-secure [ICC_IGRPEN1](#).Enable bit is a read/write alias of the ICC_MGRPEN1.EnableGrp1NS bit.

If the highest priority pending interrupt for that PE is a Group 1 interrupt using 1 of N model, then the interrupt will target another PE as a result of the Enable bit changing from 1 to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_MGRPEN1

If an interrupt is pending within the CPU interface when an Enable bit becomes 0, the interrupt must be released to allow the Distributor to forward the interrupt to a different PE.

This register is only accessible when executing in Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b110 | 0b1100 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_MGRPEN1;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b110 | 0b1100 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_MGRPEN1 = R[t];

```


ICC_MSRE, Interrupt Controller Monitor System Register Enable register

The ICC_MSRE characteristics are:

Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL3.

Configuration

AArch32 System register ICC_MSRE bits [31:0] can be mapped to AArch64 System register [ICC_SRE_EL3\[31:0\]](#), but this is not architecturally mandated.

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL3 is implemented. Otherwise, direct accesses to ICC_MSRE are UNDEFINED.

Attributes

ICC_MSRE is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|-----|-----|-----|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | Enable | | DIB | DFB | SRE | | | | | | | | | |

Bits [31:4]

Reserved, RES0.

Enable, bit [3]

Enable. Enables lower Exception level access to [ICC_SRE](#) and [ICC_HSRE](#).

| Enable | Meaning |
|--------|--|
| 0b0 | Secure EL1 accesses to Secure ICC_SRE trap to EL3. EL2 accesses to Non-secure ICC_SRE and ICC_HSRE trap to EL3. Non-secure EL1 accesses to ICC_SRE trap to EL3, unless these accesses are trapped to EL2 as a result of ICC_HSRE.Enable == 0 . |
| 0b1 | Secure EL1 accesses to Secure ICC_SRE do not trap to EL3. EL2 accesses to Non-secure ICC_SRE and ICC_HSRE do not trap to EL3. Non-secure EL1 accesses to ICC_SRE do not trap to EL3. |

If ICC_MSRE.SRE is RAO/WI, an implementation is permitted to make the Enable bit RAO/WI.

If ICC_MSRE.SRE is 0, the Enable bit behaves as 1 for all purposes other than reading the value of the bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DIB, bit [2]

Disable IRQ bypass.

| DIB | Meaning |
|-----|----------------------|
| 0b0 | IRQ bypass enabled. |
| 0b1 | IRQ bypass disabled. |

In systems that do not support IRQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

DFB, bit [1]

Disable FIQ bypass.

| DFB | Meaning |
|-----|----------------------|
| 0b0 | FIQ bypass enabled. |
| 0b1 | FIQ bypass disabled. |

In systems that do not support FIQ bypass, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

SRE, bit [0]

System Register Enable.

| SRE | Meaning |
|-----|---|
| 0b0 | The memory-mapped interface must be used. Accesses at EL3 or below to any ICH_* System register, or any EL1, EL2, or EL3 ICC_* register other than ICC_SRE , ICC_HSRE , or ICC_MSRE, are UNDEFINED. |
| 0b1 | The System register interface to the ICH_* registers and the EL1, EL2, and EL3 ICC_* registers is enabled for EL3. |

If software changes this bit from 1 to 0, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_MSRE

This register is always System register accessible.

The GIC architecture permits, but does not require, that registers can be shared between memory-mapped registers and the equivalent System registers. This means that if the memory-mapped registers have been accessed while ICC_MSRE.SRE==0, then the System registers might be modified. Therefore, software must only rely on the reset values of the System registers if there has been no use of the GIC functionality while the memory-mapped registers are in use. Otherwise, the System register values must be treated as UNKNOWN.

This register is only accessible when executing in Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b110 | 0b1100 | 0b1100 | 0b101 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return ICC_MSRE;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b110 | 0b1100 | 0b1100 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        ICC_MSRE = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICC_PMR, Interrupt Controller Interrupt Priority Mask Register

The ICC_PMR characteristics are:

Purpose

Provides an interrupt priority filter. Only interrupts with a higher priority than the value in this register are signaled to the PE.

Configuration

AArch32 System register ICC_PMR bits [31:0] are architecturally mapped to AArch64 System register [ICC_PMR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_PMR are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that writes to this register are self-synchronising. This ensures that no interrupts below the written PMR value will be taken after a write to this register is architecturally executed. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_PMR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|----------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | Priority | | | | | | | |

Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The priority mask level for the CPU interface. If the priority of an interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

The possible priority field values are as follows:

| Implemented priority bits | Possible priority field values | Number of priority levels |
|---------------------------|-------------------------------------|---------------------------|
| [7:0] | 0x00-0xFF (0-255), all values | 256 |
| [7:1] | 0x00-0xFE (0-254), even values only | 128 |
| [7:2] | 0x00-0xFC (0-252), in steps of 4 | 64 |
| [7:3] | 0x00-0xF8 (0-248), in steps of 8 | 32 |
| [7:4] | 0x00-0xF0 (0-240), in steps of 16 | 16 |

Unimplemented priority bits are RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_PMR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0100 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_PMR;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_PMR;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_PMR;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_PMR;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_PMR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_PMR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_PMR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0100 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICC_PMR = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICC_PMR = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICC_PMR = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICC_PMR = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_PMR = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_PMR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_PMR = R[t];

```


ICC_RPR, Interrupt Controller Running Priority Register

The ICC_RPR characteristics are:

Purpose

Indicates the Running priority of the CPU interface.

Configuration

AArch32 System register ICC_RPR performs the same function as AArch64 System register [ICC_RPR_EL1](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_RPR are UNDEFINED.

Attributes

ICC_RPR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|----------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | Priority | | | | | | | |

Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The current running priority on the CPU interface. This is the group priority of the current active interrupt.

The priority returned is the group priority as if the BPR for the current Exception level and Security state was set to the minimum value of BPR for the number of implemented priority bits.

Note

If 8 bits of priority are implemented the group priority is bits[7:1] of the priority.

Accessing ICC_RPR

If there are no active interrupts on the CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

Software cannot determine the number of implemented priority bits from a read of this register.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1011 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_RPR;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_RPR;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_RPR;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_RPR;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_RPR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_RPR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_RPR;

```


ICC_SGI0R, Interrupt Controller Software Generated Interrupt Group 0 Register

The ICC_SGI0R characteristics are:

Purpose

Generates Secure Group 0 SGIs.

Configuration

AArch32 System register ICC_SGI0R performs the same function as AArch64 System register [ICC_SGI0R_EL1](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_SGI0R are UNDEFINED.

Attributes

ICC_SGI0R is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|----|----|------|----|----|----|----|----|----|----|------------|----|----|----|------|----|-----|------|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | Aff3 | | | | | | | | RS | | | | RES0 | | IRM | Aff2 | | | | | | | | |
| RES0 | | | | INTID | | | | Aff1 | | | | | | | | TargetList | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:56]

Reserved, RES0.

Aff3, bits [55:48]

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

RS, bits [47:44]

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value $((RS * 16) + n)$.

When [ICC_CTLR_EL1](#).RSS==0, RS is RES0.

When [ICC_CTLR_EL1](#).RSS==1 and [GICD_TYPER](#).RSS==0, writing this register with $RS \neq 0$ is a CONSTRAINED UNPREDICTABLE choice of :

- The write is ignored.
- The RS field is treated as 0.

Bits [43:41]

Reserved, RES0.

IRM, bit [40]

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

| IRM | Meaning |
|-----|---|
| 0b0 | Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.<target list>. |
| 0b1 | Interrupts routed to all PEs in the system, excluding "self". |

Aff2, bits [39:32]

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]

Reserved, RES0.

INTID, bits [27:24]

The INTID of the SGI.

Aff1, bits [23:16]

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Note

If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

Accessing ICC_SGI0R

This register allows software executing in a Secure state to generate Group 0 SGIs. It will also allow software executing in a Non-secure state to generate Group 0 SGIs, if permitted by the settings of [GICR_NSACR](#) in the Redistributor corresponding to the target PE.

When [GICD_CTLR](#).DS==0, Non-secure writes do not generate an interrupt for a target PE if not permitted by the [GICR_NSACR](#) register associated with the target PE. For more information, see 'Use of control registers for SGI forwarding' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Note

Accesses from Secure Monitor mode are treated as Secure regardless of the value of SCR.NS.

Accesses to this register use the following encodings in the System register encoding space:

MCCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|---------------|------------|-------------|
| 0b1111 | 0b1100 | 0b0010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_SGI0R = R[t2]:R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_SGI0R = R[t2]:R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            ICC_SGI0R = R[t2]:R[t];

```


ICC_SGI1R, Interrupt Controller Software Generated Interrupt Group 1 Register

The ICC_SGI1R characteristics are:

Purpose

Generates Group 1 SGIs for the current Security state.

Configuration

AArch32 System register ICC_SGI1R performs the same function as AArch64 System register [ICC_SGI1R_EL1](#).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_SGI1R are UNDEFINED.

Under certain conditions a write to ICC_SGI1R can generate Group 0 interrupts, see 'Forwarding an SGI to a target PE' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICC_SGI1R is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|----|----|------|----|----|----|----|----|----|----|------------|----|----|----|------|----|-----|------|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | Aff3 | | | | | | | | RS | | | | RES0 | | IRM | Aff2 | | | | | | | | |
| RES0 | | | | INTID | | | | Aff1 | | | | | | | | TargetList | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:56]

Reserved, RES0.

Aff3, bits [55:48]

The affinity 3 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

RS, bits [47:44]

RangeSelector

Controls which group of 16 values is represented by the TargetList field.

TargetList[n] represents aff0 value $((RS * 16) + n)$.

When [ICC_CTLR_EL1](#).RSS==0, RS is RES0.

When [ICC_CTLR_EL1](#).RSS==1 and [GICD_TYPER](#).RSS==0, writing this register with $RS \neq 0$ is a CONSTRAINED UNPREDICTABLE choice of :

- The write is ignored.
- The RS field is treated as 0.

Bits [43:41]

Reserved, RES0.

IRM, bit [40]

Interrupt Routing Mode. Determines how the generated interrupts are distributed to PEs. Possible values are:

| IRM | Meaning |
|-----|---|
| 0b0 | Interrupts routed to the PEs specified by Aff3.Aff2.Aff1.<target list>. |
| 0b1 | Interrupts routed to all PEs in the system, excluding "self". |

Aff2, bits [39:32]

The affinity 2 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

Bits [31:28]

Reserved, RES0.

INTID, bits [27:24]

The INTID of the SGI.

Aff1, bits [23:16]

The affinity 1 value of the affinity path of the cluster for which SGI interrupts will be generated.

If the IRM bit is 1, this field is RES0.

TargetList, bits [15:0]

Target List. The set of PEs for which SGI interrupts will be generated. Each bit corresponds to the PE within a cluster with an Affinity 0 value equal to the bit number.

If a bit is 1 and the bit does not correspond to a valid target PE, the bit must be ignored by the Distributor. It is IMPLEMENTATION DEFINED whether, in such cases, a Distributor can signal a system error.

Note

If SRE is set only for Secure EL3, software executing at EL3 might use the System register interface to generate SGIs. Therefore, the Distributor must always be able to receive and acknowledge Generate SGI packets received from CPU interface regardless of the ARE settings for a Security state. However, the Distributor might discard such packets.

If the IRM bit is 1, this field is RES0.

Accessing ICC_SGI1R**Note**

Accesses from Secure Monitor mode are treated as Secure regardless of the value of SCR.NS.

Accesses to this register use the following encodings in the System register encoding space:

MCCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|---------------|------------|-------------|
| 0b1111 | 0b1100 | 0b0000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_SGI1R = R[t2]:R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_SGI1R = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_SGI1R = R[t2]:R[t];

```


ICC_SRE, Interrupt Controller System Register Enable register

The ICC_SRE characteristics are:

Purpose

Controls whether the System register interface or the memory-mapped interface to the GIC CPU interface is used for EL0 and EL1.

Configuration

AArch32 System register ICC_SRE bits [31:0] (S) are architecturally mapped to AArch64 System register [ICC_SRE_EL1\[31:0\]](#) (S).

AArch32 System register ICC_SRE bits [31:0] (NS) are architecturally mapped to AArch64 System register [ICC_SRE_EL1\[31:0\]](#) (NS).

This register is present only when AArch32 is supported and FEAT_GICv3 is implemented. Otherwise, direct accesses to ICC_SRE are UNDEFINED.

Attributes

ICC_SRE is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|-----|-----|-----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | DIB | DFB | SRE |

Bits [31:3]

Reserved, RES0.

DIB, bit [2]

Disable IRQ bypass.

| DIB | Meaning |
|-----|----------------------|
| 0b0 | IRQ bypass enabled. |
| 0b1 | IRQ bypass disabled. |

If EL3 is implemented and [GICD_CTLR](#).DS == 0, this field is a read-only alias of [ICC_MSRE](#).DIB.

If EL3 is implemented and [GICD_CTLR](#).DS == 1, and EL2 is not implemented, this field is a read-write alias of [ICC_MSRE](#).DIB.

If EL3 is not implemented and EL2 is implemented, this field is a read-only alias of [ICC_HSRE](#).DIB.

If [GICD_CTLR](#).DS == 1 and EL2 is implemented, this field is a read-only alias of [ICC_HSRE](#).DIB.

In systems that do not support IRQ bypass, this field is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

DFB, bit [1]

Disable FIQ bypass.

| DFB | Meaning |
|-----|----------------------|
| 0b0 | FIQ bypass enabled. |
| 0b1 | FIQ bypass disabled. |

If EL3 is implemented and [GICD_CTLR.DS](#) == 0, this field is a read-only alias of [ICC_MSRE.DFB](#).

If EL3 is implemented and [GICD_CTLR.DS](#) == 1, and EL2 is not implemented, this field is a read-write alias of [ICC_MSRE.DFB](#).

If EL3 is not implemented and EL2 is implemented, this field is a read-only alias of [ICC_HSRE.DFB](#).

If [GICD_CTLR.DS](#) == 1 and EL2 is implemented, this field is a read-only alias of [ICC_HSRE.DFB](#).

In systems that do not support FIQ bypass, this field is RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

SRE, bit [0]

System Register Enable.

| SRE | Meaning |
|-----|--|
| 0b0 | The memory-mapped interface must be used. Accesses at EL1 to any ICC_* System register other than ICC_SRE are UNDEFINED. |
| 0b1 | The System register interface for the current Security state is enabled. |

If software changes this bit from 1 to 0 in the Secure instance of this register, the results are UNPREDICTABLE.

If an implementation supports only a System register interface to the GIC CPU interface, this bit is RAO/WI.

If EL3 is implemented and using AArch64:

- When [ICC_SRE_EL3.SRE](#)==0 the Secure copy of this bit is RAZ/WI.
- When [ICC_SRE_EL3.SRE](#)==0 the Non-secure copy of this bit is RAZ/WI.

If EL3 is implemented and using AArch32:

- When [ICC_MSRE.SRE](#)==0 the Secure copy of this bit is RAZ/WI.
- When [ICC_MSRE.SRE](#)==0 the Non-secure copy of this bit is RAZ/WI.

If EL2 is implemented and using AArch64:

- When [ICC_SRE_EL2.SRE](#)==0 the Non-secure copy of this bit is RAZ/WI.

If EL2 is implemented and using AArch32:

- When [ICC_HSRE.SRE](#)==0 the Non-secure copy of this bit is RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICC_SRE

The GIC architecture permits, but does not require, that registers can be shared between memory-mapped registers and the equivalent System registers. This means that if the memory-mapped registers have been accessed while [ICC_SRE.SRE](#)==0, then the System registers might be modified. Therefore, software must only rely on the reset values of the System registers if there has been no use of the GIC functionality while the memory-mapped registers are in use. Otherwise, the System register values must be treated as UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICC_SRE_EL2.Enable == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICC_HSRE.Enable == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && ICC_MSRE.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_SRE_S;
        else
            return ICC_SRE_NS;
    else
        return ICC_SRE;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif ICC_MSRE.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            return ICC_SRE_S;
        else
            return ICC_SRE_NS;
    else
        return ICC_SRE;
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        return ICC_SRE_S;
    else
        return ICC_SRE_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICC_SRE_EL2.Enable == '0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICC_HSRE.Enable == '0' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && ICC_MSRE.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_SRE_S = R[t];
        else
            ICC_SRE_NS = R[t];
    else
        ICC_SRE = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && ICC_SRE_EL3.Enable == '0' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif ICC_MSRE.Enable == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) then
        if SCR_EL3.NS == '0' then
            ICC_SRE_S = R[t];
        else
            ICC_SRE_NS = R[t];
    else
        ICC_SRE = R[t];
elsif PSTATE.EL == EL3 then
    if SCR_EL3.NS == '0' then
        ICC_SRE_S = R[t];
    else
        ICC_SRE_NS = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_AP0R<n>, Interrupt Controller Hyp Active Priorities Group 0 Registers, n = 0 - 3

The ICH_AP0R<n> characteristics are:

Purpose

Provides information about Group 0 active priorities for EL2.

Configuration

AArch32 System register ICH_AP0R<n> bits [31:0] are architecturally mapped to AArch64 System register [ICH_AP0R<n>_EL2\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_AP0R<n> are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_AP0R<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

P<x>, bit [x], for x = 31 to 0

Provides the access to the virtual active priorities for Group 0 interrupts. Possible values of each bit are:

| P<x> | Meaning |
|------|---|
| 0b0 | There is no Group 0 interrupt active at the priority corresponding to that bit. |
| 0b1 | There is a Group 0 interrupt active at the priority corresponding to that bit. |

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP0R0 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP0R0 in the bits corresponding to 0:Priority[6:2].
- The active state of preemption levels 128 - 252 are held in ICH_AP0R1 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP0R0 in the bits corresponding to 00:Priority[5:1].
- The active state of preemption levels 64 - 126 are held in ICH_AP0R1 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP0R2 in the bits corresponding to 10:Priority[5:1].

- The active state of preemption levels 192 - 254 are held in ICH_AP0R3 in the bits corresponding to 11:Priority[5:1].

Note

Having the bit corresponding to a priority set to 1 in both ICH_AP0R<n> and [ICH_AP1R<n>](#) might result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICH_AP0R<n>

ICH_AP0R1 is only implemented in implementations that support 6 or more bits of preemption. ICH_AP0R2 and ICH_AP0R3 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by [ICH_VTR.PREbits](#)

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- ICH_AP0R<n>
- [ICH_AP1R<n>](#)

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|------------|
| 0b1111 | 0b100 | 0b1100 | 0b1000 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_AP0R[UInt(opc2<1:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_AP0R[UInt(opc2<1:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|------------|
| 0b1111 | 0b100 | 0b1100 | 0b1000 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_AP0R[UInt(opc2<1:0>)] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_AP0R[UInt(opc2<1:0>)] = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_AP1R<n>, Interrupt Controller Hyp Active Priorities Group 1 Registers, n = 0 - 3

The ICH_AP1R<n> characteristics are:

Purpose

Provides information about Group 1 active priorities for EL2.

Configuration

AArch32 System register ICH_AP1R<n> bits [31:0] are architecturally mapped to AArch64 System register [ICH_AP1R<n>_EL2\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_AP1R<n> are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_AP1R<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

P<x>, bit [x], for x = 31 to 0

Group 1 interrupt active priorities. Possible values of each bit are:

| P<x> | Meaning |
|------|---|
| 0b0 | There is no Group 1 interrupt active at the priority corresponding to that bit. |
| 0b1 | There is a Group 1 interrupt active at the priority corresponding to that bit. |

The correspondence between priority levels and bits depends on the number of bits of priority that are implemented.

If 5 bits of preemption are implemented (bits [7:3] of priority), then there are 32 preemption levels, and the active state of these preemption levels are held in ICH_AP1R0 in the bits corresponding to Priority[7:3].

If 6 bits of preemption are implemented (bits [7:2] of priority), then there are 64 preemption levels, and:

- The active state of preemption levels 0 - 124 are held in ICH_AP1R0 in the bits corresponding to 0:Priority[6:2].
- The active state of preemption levels 128 - 252 are held in ICH_AP1R1 in the bits corresponding to 1:Priority[6:2].

If 7 bits of preemption are implemented (bits [7:1] of priority), then there are 128 preemption levels, and:

- The active state of preemption levels 0 - 62 are held in ICH_AP1R0 in the bits corresponding to 00:Priority[5:1].
- The active state of preemption levels 64 - 126 are held in ICH_AP1R1 in the bits corresponding to 01:Priority[5:1].
- The active state of preemption levels 128 - 190 are held in ICH_AP1R2 in the bits corresponding to 10:Priority[5:1].

- The active state of preemption levels 192 - 254 are held in ICH_AP1R3 in the bits corresponding to 11:Priority[5:1].

Note

Having the bit corresponding to a priority set to 1 in both [ICH_AP0R<n>](#) and ICH_AP1R<n> might result in UNPREDICTABLE behavior of the interrupt prioritization system for virtual interrupts.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICH_AP1R<n>

ICH_AP1R1 is only implemented in implementations that support 6 or more bits of preemption. ICH_AP1R2 and ICH_AP1R3 are only implemented in implementations that support 7 bits of preemption. Unimplemented registers are UNDEFINED.

Note

The number of bits of preemption is indicated by [ICH_VTR.PREbits](#)

Writing to the active priority registers in any order other than the following order will result in UNPREDICTABLE behavior:

- [ICH_AP0R<n>](#)
- ICH_AP1R<n>

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|------------|
| 0b1111 | 0b100 | 0b1100 | 0b1001 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_AP1R[UInt(opc2<1:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_AP1R[UInt(opc2<1:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|------------|
| 0b1111 | 0b100 | 0b1100 | 0b1001 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_AP1R[UInt(opc2<1:0>)] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_AP1R[UInt(opc2<1:0>)] = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_EISR, Interrupt Controller End of Interrupt Status Register

The ICH_EISR characteristics are:

Purpose

Indicates which List registers have outstanding EOI maintenance interrupts.

Configuration

AArch32 System register ICH_EISR bits [31:0] are architecturally mapped to AArch64 System register [ICH_EISR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_EISR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_EISR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----------|----------|----------|----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | Status15 | Status14 | Status13 | Status12 | Status11 | Status10 | Status9 | Status8 | Status7 | Status6 | Status5 | Status4 | Status3 | Status2 | Status1 | Status0 |

Bits [31:16]

Reserved, RES0.

Status<n>, bit [n], for n = 15 to 0

EOI maintenance interrupt status bit for List register <n>:

| Status<n> | Meaning |
|-----------|--|
| 0b0 | List register <n>, ICH_LR<n> , does not have an EOI maintenance interrupt. |
| 0b1 | List register <n>, ICH_LR<n> , has an EOI maintenance interrupt that has not been handled. |

For any ICH_LR<n>, the corresponding status bit is set to 1 if all of the following are true:

- [ICH_LRC<n>](#).State is 0b00.
- [ICH_LRC<n>](#).HW is 0.
- [ICH_LRC<n>](#).EOI (bit [9]) is 1, indicating that when the interrupt corresponding to that List register is deactivated, a maintenance interrupt is asserted.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICH_EISR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_EISR;
elseif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_EISR;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_ELRSR, Interrupt Controller Empty List Register Status Register

The ICH_ELRSR characteristics are:

Purpose

Indicates which List registers contain valid interrupts.

Configuration

AArch32 System register ICH_ELRSR bits [31:0] are architecturally mapped to AArch64 System register [ICH_ELRSR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_ELRSR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_ELRSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----------|----------|----------|----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | Status15 | Status14 | Status13 | Status12 | Status11 | Status10 | Status9 | Status8 | Status7 | Status6 | Status5 | Status4 | Status3 | Status2 | Status1 | Status0 |

Bits [31:16]

Reserved, RES0.

Status<n>, bit [n], for n = 15 to 0

Status bit for List register <n>, [ICH_LR<n>](#):

| Status<n> | Meaning |
|-----------|--|
| 0b0 | List register ICH_LR<n> , if implemented, contains a valid interrupt. Using this List register can result in overwriting a valid interrupt. |
| 0b1 | List register ICH_LR<n> does not contain a valid interrupt. The List register is empty and can be used without overwriting a valid interrupt or losing an EOI maintenance interrupt. |

For any List register <n>, the corresponding status bit is set to 1 if [ICH_LRC<n>](#).State is 0b00 and either [ICH_LRC<n>](#).HW is 1 or [ICH_LRC<n>](#).EOI (bit [9]) is 0.

Accessing ICH_ELRSR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| | | | | |
|--------|------|-----|-----|------|
| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b101 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_ELRSR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_ELRSR;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_HCR, Interrupt Controller Hyp Control Register

The ICH_HCR characteristics are:

Purpose

Controls the environment for VMs.

Configuration

AArch32 System register ICH_HCR bits [31:0] are architecturally mapped to AArch64 System register [ICH_HCR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_HCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_HCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|------|------|-------|-------|--------|--------------|----------|----------|----------|----------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 |
| EOIcount | | | | | | | | | | | | | | | | | RES0 | TDIR | TSEI | TALL1 | TALL0 | TCRES0 | vSGIEOIcount | VGrp1DIE | VGrp1IEI | VGrp0DIE | VGrp0IEI |

EOIcount, bits [31:27]

This field is incremented whenever a successful write to a virtual EOIR or DIR register would have resulted in a virtual interrupt deactivation. That is either:

- A virtual write to EOIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when EOI mode is zero and no List Register was found.
- A virtual write to DIR with a valid interrupt identifier that is not in the LPI range (that is < 8192) when EOI mode is one and no List Register was found.

This allows software to manage more active interrupts than there are implemented List Registers.

It is CONSTRAINED UNPREDICTABLE whether a virtual write to EOIR that does not clear a bit in the Active Priorities registers ([ICH_AP0R<n>](#)/[ICH_AP1R<n>](#)) increments EOIcount. Permitted behaviors are:

- Increment EOIcount.
- Leave EOIcount unchanged.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [26:15]

Reserved, RES0.

TDIR, bit [14]

When FEAT_GICv3_TDIR is implemented:

Trap Non-secure EL1 writes to [ICC_DIR](#) and [ICV_DIR](#).

| TDIR | Meaning |
|-------------|--|
| 0b0 | Non-secure EL1 writes of ICC_DIR and ICV_DIR are not trapped to EL2, unless trapped by other mechanisms. |
| 0b1 | Non-secure EL1 writes of ICV_DIR are trapped to EL2. It is IMPLEMENTATION DEFINED whether Non-secure writes of ICC_DIR are trapped. Not trapping ICC_DIR writes is DEPRECATED. |

Arm deprecates not including this trap bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

TSEI, bit [13]

Trap all locally generated SEIs. This bit allows the hypervisor to intercept locally generated SEIs that would otherwise be taken at Non-secure EL1.

| TSEI | Meaning |
|-------------|--|
| 0b0 | Locally generated SEIs do not cause a trap to EL2. |
| 0b1 | Locally generated SEIs trap to EL2. |

If [ICH_VTR](#).SEIS is 0, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

TALL1, bit [12]

Trap all Non-secure EL1 accesses to ICC_* and ICV_* System registers for Group 1 interrupts to EL2.

| TALL1 | Meaning |
|--------------|--|
| 0b0 | Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts proceed as normal. |
| 0b1 | Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 1 interrupts trap to EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

TALL0, bit [11]

Trap all Non-secure EL1 accesses to ICC_* and ICV_* System registers for Group 0 interrupts to EL2.

| TALL0 | Meaning |
|--------------|--|
| 0b0 | Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts proceed as normal. |
| 0b1 | Non-secure EL1 accesses to ICC_* and ICV_* registers for Group 0 interrupts trap to EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

TC, bit [10]

Trap all Non-secure EL1 accesses to System registers that are common to Group 0 and Group 1 to EL2.

| TC | Meaning |
|-----|--|
| 0b0 | Non-secure EL1 accesses to common registers proceed as normal. |
| 0b1 | Non-secure EL1 accesses to common registers trap to EL2. |

This affects accesses to [ICC_SGI0R](#), [ICC_SGI1R](#), [ICC_ASGI1R](#), [ICC_CTLR](#), [ICC_DIR](#), [ICC_PMR](#), [ICC_RPR](#), [ICV_CTLR](#), [ICV_DIR](#), [ICV_PMR](#), and [ICV_RPR](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bit [9]

Reserved, RES0.

vSGIEOICount, bit [8]

When FEAT_GICv4p1 is implemented:

Controls whether deactivation of virtual SGIs can increment ICH_HCR_EL2.EOICount

| vSGIEOICount | Meaning |
|--------------|---|
| 0b0 | Deactivation of virtual SGIs can increment ICH_HCR.EOICount. |
| 0b1 | Deactivation of virtual SGIs does not increment ICH_HCR.EOICount. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

VGrp1DIE, bit [7]

VM Group 1 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is disabled:

| VGrp1DIE | Meaning |
|----------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when ICH_VMCR.VENG1 is 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp1EIE, bit [6]

VM Group 1 Enabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected vPE is enabled:

| VGrp1EIE | Meaning |
|----------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when ICH_VMCR.VENG1 is 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp0DIE, bit [5]

VM Group 0 Disabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is disabled:

| VGrp0DIE | Meaning |
|----------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when ICH_VMCR.VENG0 is 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp0EIE, bit [4]

VM Group 0 Enabled Interrupt Enable. Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected vPE is enabled:

| VGrp0EIE | Meaning |
|----------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when ICH_VMCR.VENG0 is 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

NPIE, bit [3]

No Pending Interrupt Enable. Enables the signaling of a maintenance interrupt when there are no List registers with the State field set to 0b01 (pending):

| NPIE | Meaning |
|------|---|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled while the List registers contain no interrupts in the pending state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

LRENPIE, bit [2]

List Register Entry Not Present Interrupt Enable. Enables the signaling of a maintenance interrupt while the virtual CPU interface does not have a corresponding valid List register entry for an EOI request:

| LRENPIE | Meaning |
|---------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt is asserted while the EOICount field is not 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

UIE, bit [1]

Underflow Interrupt Enable. Enables the signaling of a maintenance interrupt when the List registers are empty, or hold only one valid entry:

| UIE | Meaning |
|-----|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt is asserted if none, or only one, of the List register entries is marked as a valid interrupt. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

En, bit [0]

Enable. Global enable bit for the virtual CPU interface:

| En | Meaning |
|-----|---|
| 0b0 | Virtual CPU interface operation disabled. |
| 0b1 | Virtual CPU interface operation enabled. |

When this field is set to 0:

- The virtual CPU interface does not signal any maintenance interrupts.
- The virtual CPU interface does not signal any virtual interrupts.
- A read of [ICV_IAR0](#), [ICV_IAR1](#), [GICV_IAR](#) or [GICV_AIAR](#) returns a spurious interrupt ID.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICH_HCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_HCR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_HCR;
    
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_HCR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_HCR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_LRC<n>, Interrupt Controller List Registers, n = 0 - 15

The ICH_LRC<n> characteristics are:

Purpose

Provides interrupt context information for the virtual CPU interface.

Configuration

AArch32 System register ICH_LRC<n> bits [31:0] are architecturally mapped to AArch64 System register [ICH_LR<n>_EL2\[63:32\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_LRC<n> are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

If list register n is not implemented, then accesses to this register are UNDEFINED.

Attributes

ICH_LRC<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----|-------|----|------|----|----|----|----------|----|----|----|----|----|------|----|----|----|--------|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| State | | | HW | | Group | | RES0 | | | | Priority | | | | | | RES0 | | | | pINTID | | | | | | | | | | |

State, bits [31:30]

The state of the interrupt:

| State | Meaning |
|-------|---------------------|
| 0b00 | Invalid (Inactive). |
| 0b01 | Pending. |
| 0b10 | Active. |
| 0b11 | Pending and active. |

The GIC updates these state bits as virtual interrupts proceed through the interrupt life cycle. Entries in the invalid state are ignored, except for the purpose of generating virtual maintenance interrupts.

For hardware interrupts, the pending and active state is held in the physical Distributor rather than the virtual CPU interface. A hypervisor must only use the pending and active state for software originated interrupts, which are typically associated with virtual devices, or SGIs.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

HW, bit [29]

Indicates whether this virtual interrupt maps directly to a hardware interrupt, meaning that it corresponds to a physical interrupt. Deactivation of the virtual interrupt also causes the deactivation of the physical interrupt with the INTID that the pINTID field indicates.

| HW | Meaning |
|-----|--|
| 0b0 | The interrupt is triggered entirely by software. No notification is sent to the Distributor when the virtual interrupt is deactivated. |
| 0b1 | The interrupt maps directly to a hardware interrupt. A deactivate interrupt request is sent to the Distributor when the virtual interrupt is deactivated, using the pINTID field from this register to indicate the physical INTID. If ICH_VMCR.VEOIM is 0, this request corresponds to a write to ICC_EOIR0 or ICC_EOIR1 . Otherwise, it corresponds to a write to ICC_DIR . |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Group, bit [28]

Indicates the group for this virtual interrupt.

| Group | Meaning |
|-------|---|
| 0b0 | This is a Group 0 virtual interrupt. ICH_VMCR.VFIQEn determines whether it is signaled as a virtual IRQ or as a virtual FIQ, and ICH_VMCR.VENG0 enables signaling of this interrupt to the virtual machine. |
| 0b1 | This is a Group 1 virtual interrupt, signaled as a virtual IRQ. ICH_VMCR.VENG1 enables the signaling of this interrupt to the virtual machine. If ICH_VMCR.VCBPR is 0, then ICC_BPR1 determines if a pending Group 1 interrupt has sufficient priority to preempt current execution. Otherwise, ICH_LR<n> determines preemption. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [27:24]

Reserved, RES0.

Priority, bits [23:16]

The priority of this interrupt.

It is IMPLEMENTATION DEFINED how many bits of priority are implemented, though at least five bits must be implemented. Unimplemented bits are RES0 and start from bit[16] up to bit[18]. The number of implemented bits can be discovered from [ICH_VTR.PRIBits](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [15:13]

Reserved, RES0.

pINTID, bits [12:0]

Physical INTID, for hardware interrupts.

When ICH_LRC<n>.HW is 0 (there is no corresponding physical interrupt), this field has the following meaning:

- Bits[12:10] : RES0.

- Bit[9] : EOI. If this bit is 1, then when the interrupt identified by vINTID is deactivated, an EOI maintenance interrupt is asserted.
- Bits[8:0] : Reserved, RES0.

When ICH_LRC<n>.HW is 1 (there is a corresponding physical interrupt):

- This field indicates the physical INTID. This field is only required to implement enough bits to hold a valid value for the implemented INTID size. Any unused higher order bits are RES0.
- When [ICC_CTLR_EL1.ExtRange](#) is 0, then bits[44:42] of this field are RES0.
- If the value of pINTID is not a valid INTID, behavior is UNPREDICTABLE. If the value of pINTID indicates a PPI, this field applies to the PPI associated with this same physical PE ID as the virtual CPU interface requesting the deactivation.

A hardware physical identifier is only required in List Registers for interrupts that require deactivation. This means only 13 bits of Physical INTID are required, regardless of the number specified by [ICC_CTLR.IDbits](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICH_LRC<n>

[ICH_LR<n>](#) and ICH_LRC<n> can be updated independently.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|------------|--------|
| 0b1111 | 0b100 | 0b1100 | 0b111:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_LRC[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_LRC[UInt(CRm<0>:opc2<2:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|------------|--------|
| 0b1111 | 0b100 | 0b1100 | 0b111:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LRC[UInt(CRm<0>:opc2<2:0>)] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LRC[UInt(CRm<0>:opc2<2:0>)] = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_LR<n>, Interrupt Controller List Registers, n = 0 - 15

The ICH_LR<n> characteristics are:

Purpose

Provides interrupt context information for the virtual CPU interface.

Configuration

AArch32 System register ICH_LR<n> bits [31:0] are architecturally mapped to AArch64 System register [ICH_LR<n>_EL2\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_LR<n> are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

If list register n is not implemented, then accesses to this register are UNDEFINED.

Attributes

ICH_LR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | vINTID | | | | | | | | | | | | | | | |

vINTID, bits [31:0]

Virtual INTID of the interrupt.

If the value of vINTID is 1020-1023 and [ICH_LRC<n>.State!=0b00](#) (Inactive), behavior is UNPREDICTABLE.

Behavior is UNPREDICTABLE if two or more List Registers specify the same vINTID when:

- [ICH_LRC<n>.State == 01.](#)
- [ICH_LRC<n>.State == 10.](#)
- [ICH_LRC<n>.State == 11.](#)

It is IMPLEMENTATION DEFINED how many bits are implemented, though at least 16 bits must be implemented. Unimplemented bits are RES0. The number of implemented bits can be discovered from [ICH_VTR.IDbits](#).

Note

When a VM is using memory-mapped access to the GIC, software must ensure that the correct source PE ID is provided in bits[12:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICH_LR<n>

ICH_LR<n> and [ICH_LRC<n>](#) can be updated independently.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|------------|--------|
| 0b1111 | 0b100 | 0b1100 | 0b110:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_LR[UInt(CRm<0>:opc2<2:0>)];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_LR[UInt(CRm<0>:opc2<2:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|------------|--------|
| 0b1111 | 0b100 | 0b1100 | 0b110:n[3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LR[UInt(CRm<0>:opc2<2:0>)] = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_LR[UInt(CRm<0>:opc2<2:0>)] = R[t];

```

ICH_MISR, Interrupt Controller Maintenance Interrupt State Register

The ICH_MISR characteristics are:

Purpose

Indicates which maintenance interrupts are asserted.

Configuration

AArch32 System register ICH_MISR bits [31:0] are architecturally mapped to AArch64 System register [ICH_MISR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_MISR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_MISR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|--------|--------|--------|-----|-----|---|-----|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | VGrp1D | VGrp1E | VGrp0D | VGrp0E | NPL | REN | U | EOI | | | | | | | | |

Bits [31:8]

Reserved, RES0.

VGrp1D, bit [7]

vPE Group 1 Disabled.

| VGrp1D | Meaning |
|--------|--|
| 0b0 | vPE Group 1 Disabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 1 Disabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR.VGrp1DIE](#) is 1 and [ICH_VMCR.VENG0](#) is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp1E, bit [6]

vPE Group 1 Enabled.

| VGrp1E | Meaning |
|--------|---|
| 0b0 | vPE Group 1 Enabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 1 Enabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR.VGrp1EIE](#) is 1 and [ICH_VMCR.VENG1](#) is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp0D, bit [5]

vPE Group 0 Disabled.

| VGrp0D | Meaning |
|--------|--|
| 0b0 | vPE Group 0 Disabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 0 Disabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR.VGrp0DIE](#) is 1 and [ICH_VMCR.VENG0](#) is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp0E, bit [4]

vPE Group 0 Enabled.

| VGrp0E | Meaning |
|--------|---|
| 0b0 | vPE Group 0 Enabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 0 Enabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR.VGrp0EIE](#) is 1 and [ICH_VMCR.VENG0](#) is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

NP, bit [3]

No Pending.

| NP | Meaning |
|-----|--|
| 0b0 | No Pending maintenance interrupt not asserted. |
| 0b1 | No Pending maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR.NPIE](#) is 1 and no List register is in pending state.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

LREN, bit [2]

List Register Entry Not Present.

| LREN | Meaning |
|------|---|
| 0b0 | List Register Entry Not Present maintenance interrupt not asserted. |
| 0b1 | List Register Entry Not Present maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR.LRENPIE](#) is 1 and [ICH_HCR.EOIcount](#) is non-zero.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

U, bit [1]

Underflow.

| U | Meaning |
|-----|---|
| 0b0 | Underflow maintenance interrupt not asserted. |
| 0b1 | Underflow maintenance interrupt asserted. |

This maintenance interrupt is asserted when [ICH_HCR](#).UIE is 1 and zero or one of the List register entries are marked as a valid interrupt, that is, if the corresponding [ICH_LRC<n>](#).State bits do not equal 0x0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

EOI, bit [0]

End Of Interrupt.

| EOI | Meaning |
|-----|--|
| 0b0 | End Of Interrupt maintenance interrupt not asserted. |
| 0b1 | End Of Interrupt maintenance interrupt asserted. |

This maintenance interrupt is asserted when at least one bit in [ICH_EISR](#) is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICH_MISR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_MISR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_MISR;

```

ICH_VMCR, Interrupt Controller Virtual Machine Control Register

The ICH_VMCR characteristics are:

Purpose

Enables the hypervisor to save and restore the virtual machine view of the GIC state.

Configuration

AArch32 System register ICH_VMCR bits [31:0] are architecturally mapped to AArch64 System register [ICH_VMCR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_VMCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

ICH_VMCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|-------|------|----|----|----|-------|------|-------|--------|---------|-------|-------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VPMR | | | | | | | | VBPR0 | VBPR1 | RES0 | | | | VEOIM | RES0 | VCBPR | VFIQEn | VackCtI | VENG1 | VENG0 | | | | | | | | | | | |

VPMR, bits [31:24]

Virtual Priority Mask. The priority mask level for the virtual CPU interface. If the priority of a pending virtual interrupt is higher than the value indicated by this field, the interface signals the virtual interrupt to the PE.

This field is an alias of [ICV_PMR](#).Priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VBPR0, bits [23:21]

Virtual Binary Point Register, Group 0. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption, and also determines Group 1 interrupt preemption if ICH_VMCR.VCBPR == 1.

This field is an alias of [ICV_BPR0](#).BinaryPoint.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VBPR1, bits [20:18]

Virtual Binary Point Register, Group 1. Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption if [ICH_VMCR](#).VCBPR == 0.

This field is an alias of [ICV_BPR1](#).BinaryPoint.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [17:10]

Reserved, RES0.

VEOIM, bit [9]

Virtual EOI mode. Controls whether a write to an End of Interrupt register also deactivates the virtual interrupt:

| VEOIM | Meaning |
|-------|---|
| 0b0 | ICV_EOIR0 and ICV_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR are UNPREDICTABLE. |
| 0b1 | ICV_EOIR0 and ICV_EOIR1 provide priority drop functionality only. ICV_DIR provides interrupt deactivation functionality. |

This bit is an alias of [ICV_CTLR](#).EOImode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:5]

Reserved, RES0.

VCBPR, bit [4]

Virtual Common Binary Point Register. Possible values of this bit are:

| VCBPR | Meaning |
|-------|--|
| 0b0 | ICV_BPR0 determines the preemption group for virtual Group 0 interrupts only. ICV_BPR1 determines the preemption group for virtual Group 1 interrupts. |
| 0b1 | ICV_BPR0 determines the preemption group for both virtual Group 0 and virtual Group 1 interrupts. Reads of ICV_BPR1 return ICV_BPR0 plus one, saturated to 0b111. Writes to ICV_BPR1 are ignored. |

This field is an alias of [ICV_CTLR](#).CBPR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VFIQEn, bit [3]

Virtual FIQ enable. Possible values of this bit are:

| VFIQEn | Meaning |
|--------|---|
| 0b0 | Group 0 virtual interrupts are presented as virtual IRQs. |
| 0b1 | Group 0 virtual interrupts are presented as virtual FIQs. |

This bit is an alias of [GICV_CTLR](#).FIQEn.

In implementations where the Non-secure copy of [ICC_SRE](#).SRE is always 1, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VAckCtl, bit [2]

Virtual AckCtl. Possible values of this bit are:

| VAckCtl | Meaning |
|---------|--|
| 0b0 | If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns an INTID of 1022. |
| 0b1 | If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns the INTID of the corresponding interrupt. |

This bit is an alias of [GICV_CTLR](#).AckCtl.

This field is supported for backwards compatibility with GICv2. Arm deprecates the use of this field.

In implementations where the Non-secure copy of [ICC_SRE](#).SRE is always 1, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VENG1, bit [1]

Virtual Group 1 interrupt enable. Possible values of this bit are:

| VENG1 | Meaning |
|-------|--|
| 0b0 | Virtual Group 1 interrupts are disabled. |
| 0b1 | Virtual Group 1 interrupts are enabled. |

This bit is an alias of [ICV_IGRPEN1](#).Enable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VENG0, bit [0]

Virtual Group 0 interrupt enable. Possible values of this bit are:

| VENG0 | Meaning |
|-------|--|
| 0b0 | Virtual Group 0 interrupts are disabled. |
| 0b1 | Virtual Group 0 interrupts are enabled. |

This bit is an alias of [ICV_IGRPEN0](#).Enable.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICH_VMCR

When EL2 is using System register access, EL1 using either System register or memory-mapped access must be supported.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_VMCR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_VMCR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_VMCR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICH_VMCR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICH_VTR, Interrupt Controller VGIC Type Register

The ICH_VTR characteristics are:

Purpose

Reports supported GIC virtualization features.

Configuration

AArch32 System register ICH_VTR bits [31:0] are architecturally mapped to AArch64 System register [ICH_VTR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and (EL2 is implemented or EL3 is implemented). Otherwise, direct accesses to ICH_VTR are UNDEFINED.

If EL2 is not implemented, all bits in this register are RES0 from EL3, except for nV4, which is RES1 from EL3.

Attributes

ICH_VTR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|---------|----|----|--------|----|----|---------------|----|----|------|----|----|----|----|----|----|----|----|----|---|---|----------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PRIbits | | | PREbits | | | IDbits | | | SEISA3VnV4TDS | | | RES0 | | | | | | | | | | | | ListRegs | | | | | | | |

PRibits, bits [31:29]

Priority bits. The number of virtual priority bits implemented, minus one.

An implementation must implement at least 32 levels of virtual priority (5 priority bits).

This field is an alias of [ICV_CTLR.PRibits](#).

PREbits, bits [28:26]

The number of virtual preemption bits implemented, minus one.

An implementation must implement at least 32 levels of virtual preemption priority (5 preemption bits).

The value of this field must be less than or equal to the value of ICH_VTR.PRibits.

IDbits, bits [25:23]

The number of virtual interrupt identifier bits supported:

| IDbits | Meaning |
|--------|----------|
| 0b000 | 16 bits. |
| 0b001 | 24 bits. |

All other values are reserved.

This field is an alias of [ICV_CTLR.IDbits](#).

SEIS, bit [22]

SEI Support. Indicates whether the virtual CPU interface supports generation of SEIs:

| SEIS | Meaning |
|------|--|
| 0b0 | The virtual CPU interface logic does not support generation of SEIs. |
| 0b1 | The virtual CPU interface logic supports generation of SEIs. |

This bit is an alias of [ICV_CTLR](#).SEIS.

A3V, bit [21]

Affinity 3 Valid. Possible values are:

| A3V | Meaning |
|-----|---|
| 0b0 | The virtual CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers. |
| 0b1 | The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers. |

This bit is an alias of [ICV_CTLR](#).A3V.

nV4, bit [20]

Direct injection of virtual interrupts not supported. Possible values are:

| nV4 | Meaning |
|-----|--|
| 0b0 | The CPU interface logic supports direct injection of virtual interrupts. |
| 0b1 | The CPU interface logic does not support direct injection of virtual interrupts. |

In GICv3, the only permitted value is 0b1.

TDS, bit [19]

Separate trapping of Non-secure EL1 writes to [ICV_DIR](#) supported.

| TDS | Meaning |
|-----|--|
| 0b0 | Implementation does not support ICH_HCR .TDIR. |
| 0b1 | Implementation supports ICH_HCR .TDIR. |

FEAT_GICv3_TDIR implements the functionality added by the value 0b1.

Bits [18:5]

Reserved, RES0.

ListRegs, bits [4:0]

The number of implemented List registers, minus one. For example, a value of 0b01111 indicates that the maximum of 16 List registers are implemented.

Accessing ICH_VTR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if ICC_HSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_VTR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICH_VTR;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICIALLU, Instruction Cache Invalidate All to PoU

The ICIALLU characteristics are:

Purpose

Invalidate all instruction caches of the PE executing the instruction to the Point of Unification. If branch predictors are architecturally visible, also flush branch predictors.

Configuration

AArch32 System instruction ICIALLU performs the same function as AArch64 System instruction [IC IALLU](#).

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ICIALLU are UNDEFINED.

Attributes

ICIALLU is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the ICIALLU instruction

The PE ignores the value of <Rt>. Software does not have to write a value to this register before issuing this instruction.

When [HCR.FB](#) is 1, at Non-secure EL1 this instruction executes as a [ICIALLUIS](#).

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TOCU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TPU == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TOCU == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
        AArch32.IC(CacheOpScope_ALLUIS);
    else
        AArch32.IC(CacheOpScope_ALLU);
elsif PSTATE.EL == EL2 then
    AArch32.IC(CacheOpScope_ALLU);
elsif PSTATE.EL == EL3 then
    AArch32.IC(CacheOpScope_ALLU);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICIALLUIS, Instruction Cache Invalidate All to PoU, Inner Shareable

The ICIALLUIS characteristics are:

Purpose

Invalidate all instruction caches in the Inner Shareable domain of the PE executing the instruction to the Point of Unification. If branch predictors are architecturally visible, also flush branch predictors.

Configuration

AArch32 System instruction ICIALLUIS performs the same function as AArch64 System instruction [IC IALLUIS](#).

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ICIALLUIS are UNDEFINED.

Attributes

ICIALLUIS is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the ICIALLUIS instruction

The PE ignores the value of <Rt>. Software does not have to write a value to this register before issuing this instruction.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TICAB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TPU == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TICAB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.IC(CacheOpScope_ALLUIS);
    endif PSTATE.EL == EL2 then
        AArch32.IC(CacheOpScope_ALLUIS);
    elsif PSTATE.EL == EL3 then
        AArch32.IC(CacheOpScope_ALLUIS);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICIMVAU, Instruction Cache line Invalidate by VA to PoU

The ICIMVAU characteristics are:

Purpose

Invalidate instruction cache line by virtual address to PoU.

Configuration

AArch32 System instruction ICIMVAU performs the same function as AArch64 System instruction [IC IVAU](#).

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ICIMVAU are UNDEFINED.

Attributes

ICIMVAU is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Virtual address to use | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Virtual address to use. No alignment restrictions apply to this VA.

Executing the ICIMVAU instruction

Execution of this instruction might require an address translation from VA to PA, and that translation might fault.

If FEAT_CMOW is implemented, [HCRX_EL2](#).CMOW is 1, and EL1 or EL0 access is enabled, when executed at EL1 or EL0, the instruction has stage 2 read permission to the VA but does not have stage 2 write permission to the VA, the instruction generates a stage 2 Permission fault.

For more information, see 'AArch32 instruction cache maintenance instructions (IC*)'.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TPU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TOCU == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TPU == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TOCU == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        AArch32.IC(R[t], CacheOpScope_PoU);
elsif PSTATE.EL == EL2 then
    AArch32.IC(R[t], CacheOpScope_PoU);
elsif PSTATE.EL == EL3 then
    AArch32.IC(R[t], CacheOpScope_PoU);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_AP0R<n>, Interrupt Controller Virtual Active Priorities Group 0 Registers, n = 0 - 3

The ICV_AP0R<n> characteristics are:

Purpose

Provides information about virtual Group 0 active priorities.

Configuration

AArch32 System register ICV_AP0R<n> bits [31:0] are architecturally mapped to AArch64 System register [ICV_AP0R<n>_EL1\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_AP0R<n> are UNDEFINED.

Attributes

ICV_AP0R<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

Accessing ICV_AP0R<n>

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 0 active priorities) might result in UNPREDICTABLE behavior of the virtual interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICV_AP0R1 is only implemented in implementations that support 6 or more bits of priority. ICV_AP0R2 and ICV_AP0R3 are only implemented in implementations that support 7 bits of priority. Unimplemented registers are UNDEFINED.

Writing to the active priority registers in any order other than the following order might result in UNPREDICTABLE behavior of the interrupt prioritization system:

- ICV_AP0R<n>.
- [ICV_AP1R<n>](#).

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|------------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b1:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_AP0R[UInt(opc2<1:0>)];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_AP0R[UInt(opc2<1:0>)];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_AP0R[UInt(opc2<1:0>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_AP0R[UInt(opc2<1:0>)];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            return ICC_AP0R[UInt(opc2<1:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|------------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b1:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_AP0R[UInt(opc2<1:0>)] = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_AP0R[UInt(opc2<1:0>)] = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_AP0R[UInt(opc2<1:0>)] = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_AP0R[UInt(opc2<1:0>)] = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            ICC_AP0R[UInt(opc2<1:0>)] = R[t];

```


ICV_AP1R<n>, Interrupt Controller Virtual Active Priorities Group 1 Registers, n = 0 - 3

The ICV_AP1R<n> characteristics are:

Purpose

Provides information about virtual Group 1 active priorities.

Configuration

AArch32 System register ICV_AP1R<n> bits [31:0] are architecturally mapped to AArch64 System register [ICV_AP1R<n>_EL1\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_AP1R<n> are UNDEFINED.

Attributes

ICV_AP1R<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

Accessing ICV_AP1R<n>

Writing to these registers with any value other than the last read value of the register (or 0x00000000 when there are no Group 1 active priorities) might result in UNPREDICTABLE behavior of the virtual interrupt prioritization system, causing:

- Interrupts that should preempt execution to not preempt execution.
- Interrupts that should not preempt execution to preempt execution.

ICV_AP1R1 is only implemented in implementations that support 6 or more bits of priority. ICV_AP1R2 and ICV_AP1R3 are only implemented in implementations that support 7 bits of priority. Unimplemented registers are UNDEFINED.

Writing to the active priority registers in any order other than the following order might result in UNPREDICTABLE behavior of the interrupt prioritization system:

- [ICV_AP0R<n>](#).
- ICV_AP1R<n>.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1100 | 0b1001 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_AP1R[UInt(opc2<1:0>)];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_AP1R[UInt(opc2<1:0>)];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            return ICC_AP1R_NS[UInt(opc2<1:0>)];
        else
            return ICC_AP1R[UInt(opc2<1:0>)];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            return ICC_AP1R_NS[UInt(opc2<1:0>)];
        else
            return ICC_AP1R[UInt(opc2<1:0>)];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            if SCR.NS == '0' then
                return ICC_AP1R_S[UInt(opc2<1:0>)];
            else
                return ICC_AP1R_NS[UInt(opc2<1:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1100 | 0b1001 | 0b0:n[1:0] |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_AP1R[UInt(opc2<1:0>)] = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_AP1R[UInt(opc2<1:0>)] = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            ICC_AP1R_NS[UInt(opc2<1:0>)] = R[t];
        else
            ICC_AP1R[UInt(opc2<1:0>)] = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            ICC_AP1R_NS[UInt(opc2<1:0>)] = R[t];
        else
            ICC_AP1R[UInt(opc2<1:0>)] = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            if SCR.NS == '0' then
                ICC_AP1R_S[UInt(opc2<1:0>)] = R[t];
            else
                ICC_AP1R_NS[UInt(opc2<1:0>)] = R[t];

```


ICV_BPR0, Interrupt Controller Virtual Binary Point Register 0

The ICV_BPR0 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 0 interrupt preemption.

Configuration

AArch32 System register ICV_BPR0 bits [31:0] are architecturally mapped to AArch64 System register [ICV_BPR0_EL1\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_BPR0 are UNDEFINED.

Attributes

ICV_BPR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | BinaryPoint | | | | | | | | | | | | | |

Bits [31:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

The value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. This is done as follows:

| Binary point value | Group priority field | Subpriority field | Field with binary point |
|--------------------|----------------------|-------------------|-------------------------|
| 0 | [7:1] | [0] | ggggggg.s |
| 1 | [7:2] | [1:0] | gggggg.ss |
| 2 | [7:3] | [2:0] | ggggg.sss |
| 3 | [7:4] | [3:0] | gggg.ssss |
| 4 | [7:5] | [4:0] | ggg.sssss |
| 5 | [7:6] | [5:0] | gg.ssssss |
| 6 | [7] | [6:0] | g.sssssss |
| 7 | No preemption | [7:0] | .ssssssss |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_BPR0

The minimum binary point value is derived from the number of implemented priority bits. The number of priority bits is IMPLEMENTATION DEFINED, and reported by [ICV_CTLR.PRIbits](#).

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is set to the minimum supported value.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_BPR0;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_BPR0;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_BPR0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_BPR0;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_BPR0;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_BPR0 = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_BPR0 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_BPR0 = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_BPR0 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_BPR0 = R[t];

```

ICV_BPR1, Interrupt Controller Virtual Binary Point Register 1

The ICV_BPR1 characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines virtual Group 1 interrupt preemption.

Configuration

AArch32 System register ICV_BPR1 bits [31:0] are architecturally mapped to AArch64 System register [ICV_BPR1_EL1\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_BPR1 are UNDEFINED.

Attributes

ICV_BPR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | BinaryPoint | | | | | | | | | | | | | |

Bits [31:3]

Reserved, RES0.

BinaryPoint, bits [2:0]

If the GIC is configured to use separate binary point fields for Group 0 and Group 1 interrupts, the value of this field controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

For more information about priorities, see 'Priority grouping' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

An attempt to program this field to a value less than the minimum value sets the field to the minimum value.

If [ICV_CTLR](#).CBPR is set to 1, Non-secure EL1 reads return [ICV_BPR0](#) + 1 saturated to 0b111. Non-secure EL1 writes are ignored.

If [ICV_CTLR](#).CBPR is set to 1, Secure EL1 reads return [ICV_BPR0](#). Secure EL1 writes modify [ICV_BPR0](#)

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_BPR1

The minimum value of this register is equal to the minimum value of [ICV_BPR0](#) plus one.

An attempt to program the binary point field to a value less than the minimum value sets the field to the minimum value. On a reset, the binary point field is UNKNOWN.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_BPR1;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_BPR1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        return ICC_BPR1_NS;
    else
        return ICC_BPR1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        return ICC_BPR1_NS;
    else
        return ICC_BPR1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            return ICC_BPR1_S;
        else
            return ICC_BPR1_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_BPR1 = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_BPR1 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            ICC_BPR1_NS = R[t];
        else
            ICC_BPR1 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            ICC_BPR1_NS = R[t];
        else
            ICC_BPR1 = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            if SCR.NS == '0' then
                ICC_BPR1_S = R[t];
            else
                ICC_BPR1_NS = R[t];

```


ICV_CTLR, Interrupt Controller Virtual Control Register

The ICV_CTLR characteristics are:

Purpose

Controls aspects of the behavior of the GIC virtual CPU interface and provides information about the features implemented.

Configuration

AArch32 System register ICV_CTLR bits [31:0] are architecturally mapped to AArch64 System register [ICV_CTLR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_CTLR are UNDEFINED.

Attributes

ICV_CTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----------|-----|------|-----|------|--------|---------|------|----|----|---|---------|------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | ExtRange | RSS | RES0 | A3V | SEIS | IDbits | PRIbits | RES0 | | | | EOImode | CBPR | | | | | | | |

Bits [31:20]

Reserved, RES0.

ExtRange, bit [19]

Extended INTID range (read-only).

| ExtRange | Meaning |
|----------|---|
| 0b0 | <p>CPU interface does not support INTIDs in the range 1024..8191. Behaviour is UNPREDICTABLE if the IRI delivers an interrupt in the range 1024 to 8191 to the CPU interface.</p> <hr/> <p>Note Arm strongly recommends that the IRI is not configured to deliver interrupts in this range to a PE that does not support them.</p> |
| 0b1 | <p>CPU interface supports INTIDs in the range 1024..8191. All INTIDs in the range 1024..8191 are treated as requiring deactivation.</p> |

ICV_CTLR.ExtRange is an alias of [ICC_CTLR](#).ExtRange.

RSS, bit [18]

Range Selector Support. Possible values are:

| RSS | Meaning |
|------------|--|
| 0b0 | Targeted SGIs with affinity level 0 values of 0 - 15 are supported. |
| 0b1 | Targeted SGIs with affinity level 0 values of 0 - 255 are supported. |

This bit is read-only.

Bits [17:16]

Reserved, RES0.

A3V, bit [15]

Affinity 3 Valid. Read-only and writes are ignored. Possible values are:

| A3V | Meaning |
|------------|---|
| 0b0 | The virtual CPU interface logic only supports zero values of Affinity 3 in SGI generation System registers. |
| 0b1 | The virtual CPU interface logic supports non-zero values of Affinity 3 in SGI generation System registers. |

SEIS, bit [14]

SEI Support. Read-only and writes are ignored. Indicates whether the virtual CPU interface supports local generation of SEIs:

| SEIS | Meaning |
|-------------|--|
| 0b0 | The virtual CPU interface logic does not support local generation of SEIs. |
| 0b1 | The virtual CPU interface logic supports local generation of SEIs. |

IDbits, bits [13:11]

Identifier bits. Read-only and writes are ignored. The number of virtual interrupt identifier bits supported:

| IDbits | Meaning |
|---------------|----------------|
| 0b000 | 16 bits. |
| 0b001 | 24 bits. |

All other values are reserved.

PRbits, bits [10:8]

Priority bits. Read-only and writes are ignored. The number of priority bits implemented, minus one.

An implementation must implement at least 32 levels of physical priority (5 priority bits).

Note

This field always returns the number of priority bits implemented.

The division between group priority and subpriority is defined in the binary point registers [ICV_BPR0](#) and [ICV_BPR1](#).

Bits [7:2]

Reserved, RES0.

EOImode, bit [1]

Virtual EOI mode. Controls whether a write to an End of Interrupt register also deactivates the virtual interrupt:

| EOImode | Meaning |
|---------|---|
| 0b0 | ICV_EOIR0 and ICV_EOIR1 provide both priority drop and interrupt deactivation functionality. Accesses to ICV_DIR are UNPREDICTABLE. |
| 0b1 | ICV_EOIR0 and ICV_EOIR1 provide priority drop functionality only. ICV_DIR provides interrupt deactivation functionality. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CBPR, bit [0]

Common Binary Point Register. Controls whether the same register is used for interrupt preemption of both virtual Group 0 and virtual Group 1 interrupts:

| CBPR | Meaning |
|------|---|
| 0b0 | ICV_BPR0 determines the preemption group for virtual Group 0 interrupts only. ICV_BPR1 determines the preemption group for virtual Group 1 interrupts. |
| 0b1 | Non-secure reads of ICV_BPR1 return ICV_BPR0 plus one, saturated to 0b111. Non-secure writes to ICV_BPR1 are ignored. Secure reads of ICV_BPR1 return ICV_BPR0 . Secure writes of ICV_BPR1 modify ICV_BPR0 . |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing ICV_CTLR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_CTLR;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_CTLR;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_CTLR;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_CTLR;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        return ICC_CTLR_NS;
    else
        return ICC_CTLR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    elsif HaveEL(EL3) then
        return ICC_CTLR_NS;
    else
        return ICC_CTLR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            return ICC_CTLR_S;

```

```
else
    return ICC_CTLR_NS;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_CTLR = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_CTLR = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_CTLR = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_CTLR = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
            elsif HaveEL(EL3) then
                ICC_CTLR_NS = R[t];
            else
                ICC_CTLR = R[t];
        elsif PSTATE.EL == EL2 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
                UNDEFINED;
            elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
                UNDEFINED;
            elsif ICC_HSRE.SRE == '0' then
                UNDEFINED;
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
                elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
                    if Halted() && EDSCR.SDD == '1' then
                        UNDEFINED;
                    else
                        AArch32.TakeMonitorTrapException();
                    elsif HaveEL(EL3) then
                        ICC_CTLR_NS = R[t];
                    else
                        ICC_CTLR = R[t];
        elsif PSTATE.EL == EL3 then
            if ICC_MSRE.SRE == '0' then
                UNDEFINED;
            else
                if SCR.NS == '0' then
                    ICC_CTLR_S = R[t];

```

```
else  
    ICC_CTLR_NS = R[t];
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ICV_DIR, Interrupt Controller Deactivate Virtual Interrupt Register

The ICV_DIR characteristics are:

Purpose

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified virtual interrupt.

Configuration

AArch32 System register ICV_DIR bits [31:0] performs the same function as AArch64 System register [ICV_DIR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_DIR are UNDEFINED.

Attributes

ICV_DIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the virtual interrupt to be deactivated.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_DIR

When EOImode == 0, writes are ignored. In systems supporting system error generation, an implementation might generate an SEI.

Accesses to this register use the following encodings in the System register encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TDIR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TDIR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_DIR = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_DIR = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_DIR = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_DIR = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
            else
                ICC_DIR = R[t];
        elsif PSTATE.EL == EL2 then
            if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
                UNDEFINED;
            elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
                UNDEFINED;
            elsif ICC_HSRE.SRE == '0' then
                UNDEFINED;
            elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
                if Halted() && EDSCR.SDD == '1' then
                    UNDEFINED;
                else
                    AArch32.TakeMonitorTrapException();
            else
                ICC_DIR = R[t];
        elsif PSTATE.EL == EL3 then
            if ICC_MSRE.SRE == '0' then
                UNDEFINED;
            else
                ICC_DIR = R[t];

```


ICV_EOIR0, Interrupt Controller Virtual End Of Interrupt Register 0

The ICV_EOIR0 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified virtual Group 0 interrupt.

Configuration

AArch32 System register ICV_EOIR0 performs the same function as AArch64 System register [ICV_EOIR0_EL1](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_EOIR0 are UNDEFINED.

Attributes

ICV_EOIR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding [ICV_IAR0](#) access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR](#).IDbits. If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the [ICV_CTLR](#).EOImode bit is 0, a write to this register drops the priority for the virtual interrupt, and also deactivates the virtual interrupt.

If the [ICV_CTLR](#).EOImode bit is 1, a write to this register only drops the priority for the virtual interrupt. Software must write to [ICV_DIR](#) to deactivate the virtual interrupt.

Accessing ICV_EOIR0

A write to this register must correspond to the most recent valid read by this vPE from a Virtual Interrupt Acknowledge Register, and must correspond to the INTID that was read from [ICV_IAR0](#), otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

Accesses to this register use the following encodings in the System register encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_EOIR0 = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_EOIR0 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_EOIR0 = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_EOIR0 = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_EOIR0 = R[t];

```

ICV_EOIR1, Interrupt Controller Virtual End Of Interrupt Register 1

The ICV_EOIR1 characteristics are:

Purpose

A PE writes to this register to inform the CPU interface that it has completed the processing of the specified virtual Group 1 interrupt.

Configuration

AArch32 System register ICV_EOIR1 performs the same function as AArch64 System register [ICV_EOIR1_EL1](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_EOIR1 are UNDEFINED.

Attributes

ICV_EOIR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID from the corresponding [ICV_IAR1](#) access.

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

If the [ICV_CTLR.EOImode](#) bit is 0, a write to this register drops the priority for the virtual interrupt, and also deactivates the virtual interrupt.

If the [ICV_CTLR.EOImode](#) bit is 1, a write to this register only drops the priority for the virtual interrupt. Software must write to [ICV_DIR](#) to deactivate the virtual interrupt.

Accessing ICV_EOIR1

A write to this register must correspond to the most recent valid read by this vPE from a Virtual Interrupt Acknowledge Register, and must correspond to the INTID that was read from [ICV_IAR1](#), otherwise the system behavior is UNPREDICTABLE. A valid read is a read that returns a valid INTID that is not a special INTID.

Accesses to this register use the following encodings in the System register encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_EOIR1 = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_EOIR1 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_EOIR1 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_EOIR1 = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            ICC_EOIR1 = R[t];

```

ICV_HPPIR0, Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0

The ICV_HPPIR0 characteristics are:

Purpose

Indicates the highest priority pending virtual Group 0 interrupt on the virtual CPU interface.

Configuration

AArch32 System register ICV_HPPIR0 performs the same function as AArch64 System register [ICV_HPPIR0_EL1](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_HPPIR0 are UNDEFINED.

Attributes

ICV_HPPIR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending virtual interrupt.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_HPPIR0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_HPPIR0;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_HPPIR0;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_HPPIR0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_HPPIR0;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_HPPIR0;

```

ICV_HPPIR1, Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1

The ICV_HPPIR1 characteristics are:

Purpose

Indicates the highest priority pending virtual Group 1 interrupt on the virtual CPU interface.

Configuration

AArch32 System register ICV_HPPIR1 performs the same function as AArch64 System register [ICV_HPPIR1_EL1](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_HPPIR1 are UNDEFINED.

Attributes

ICV_HPPIR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the highest priority pending virtual interrupt.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_HPPIR1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_HPPIR1;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_HPPIR1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_HPPIR1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_HPPIR1;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_HPPIR1;

```

ICV_IAR0, Interrupt Controller Virtual Interrupt Acknowledge Register 0

The ICV_IAR0 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled virtual Group 0 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch32 System register ICV_IAR0 performs the same function as AArch64 System register [ICV_IAR0_EL1](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_IAR0 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when $PSTATE.\{I,F\} == \{0,0\}$). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_IAR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_IAR0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_IAR0;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_IAR0;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_IAR0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_IAR0;
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            return ICC_IAR0;

```


ICV_IAR1, Interrupt Controller Virtual Interrupt Acknowledge Register 1

The ICV_IAR1 characteristics are:

Purpose

The PE reads this register to obtain the INTID of the signaled virtual Group 1 interrupt. This read acts as an acknowledge for the interrupt.

Configuration

AArch32 System register ICV_IAR1 performs the same function as AArch64 System register [ICV_IAR1_EL1](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_IAR1 are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that reads of this register are self-synchronising when interrupts are masked by the PE (that is when $PSTATE.\{I,F\} == \{0,0\}$). This ensures that the effect of activating an interrupt on the signaling of interrupt exceptions is observed when a read of this register is architecturally executed so that no spurious interrupt exception occurs if interrupts are unmasked by an instruction immediately following the read. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_IAR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled virtual interrupt.

This is the INTID of the highest priority pending virtual interrupt, if that interrupt is of sufficient priority for it to be signaled to the PE, and if it can be acknowledged.

If the highest priority pending interrupt is not observable, this field contains a special INTID to indicate the reason. This special INTID can take the value 1023 only. For more information, see 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field has either 16 or 24 bits implemented. The number of implemented bits can be found in [ICV_CTLR.IDbits](#). If only 16 bits are implemented, bits [23:16] of this register are RES0.

Accessing ICV_IAR1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_IAR1;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_IAR1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_IAR1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_IAR1;
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            return ICC_IAR1;

```


ICV_IGRPEN0, Interrupt Controller Virtual Interrupt Group 0 Enable register

The ICV_IGRPEN0 characteristics are:

Purpose

Controls whether virtual Group 0 interrupts are enabled or not.

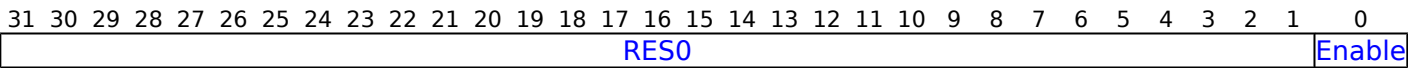
Configuration

AArch32 System register ICV_IGRPEN0 bits [31:0] are architecturally mapped to AArch64 System register [ICV_IGRPEN0_EL1\[31:0\]](#).
This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_IGRPEN0 are UNDEFINED.

Attributes

ICV_IGRPEN0 is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

Enable, bit [0]

Enables virtual Group 0 interrupts.

| Enable | Meaning |
|--------|--|
| 0b0 | Virtual Group 0 interrupts are disabled. |
| 0b1 | Virtual Group 0 interrupts are enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICV_IGRPEN0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_IGRPEN0;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_IGRPEN0;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_IGRPEN0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_IGRPEN0;
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            return ICC_IGRPEN0;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_IGRPEN0 = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_IGRPEN0 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_IGRPEN0 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.FIQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            ICC_IGRPEN0 = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            ICC_IGRPEN0 = R[t];

```

ICV_IGRPEN1, Interrupt Controller Virtual Interrupt Group 1 Enable register

The ICV_IGRPEN1 characteristics are:

Purpose

Controls whether virtual Group 1 interrupts are enabled for the current Security state.

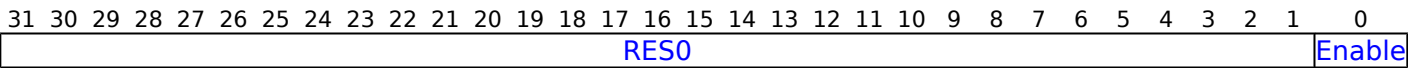
Configuration

AArch32 System register ICV_IGRPEN1 bits [31:0] are architecturally mapped to AArch64 System register [ICV_IGRPEN1_EL1\[31:0\]](#).
This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_IGRPEN1 are UNDEFINED.

Attributes

ICV_IGRPEN1 is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

Enable, bit [0]

Enables virtual Group 1 interrupts.

| Enable | Meaning |
|--------|--|
| 0b0 | Virtual Group 1 interrupts are disabled. |
| 0b1 | Virtual Group 1 interrupts are enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICV_IGRPEN1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_IGRPEN1;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_IGRPEN1;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            return ICC_IGRPEN1_NS;
        else
            return ICC_IGRPEN1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            return ICC_IGRPEN1_NS;
        else
            return ICC_IGRPEN1;
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            if SCR.NS == '0' then
                return ICC_IGRPEN1_S;
            else
                return ICC_IGRPEN1_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif ICC_SRE.SRE == '0' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TALL1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TALL1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_IGRPEN1 = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_IGRPEN1 = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            ICC_IGRPEN1_NS = R[t];
        else
            ICC_IGRPEN1 = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.IRQ == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        elsif HaveEL(EL3) then
            ICC_IGRPEN1_NS = R[t];
        else
            ICC_IGRPEN1 = R[t];
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            if SCR.NS == '0' then
                ICC_IGRPEN1_S = R[t];
            else
                ICC_IGRPEN1_NS = R[t];

```


ICV_PMR, Interrupt Controller Virtual Interrupt Priority Mask Register

The ICV_PMR characteristics are:

Purpose

Provides a virtual interrupt priority filter. Only virtual interrupts with a higher priority than the value in this register are signaled to the PE.

Configuration

AArch32 System register ICV_PMR bits [31:0] are architecturally mapped to AArch64 System register [ICV_PMR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_PMR are UNDEFINED.

To allow software to ensure appropriate observability of actions initiated by GIC register accesses, the PE and CPU interface logic must ensure that writes to this register are self-synchronising. This ensures that no interrupts below the written PMR value will be taken after a write to this register is architecturally executed. For more information, see 'Observability of the effects of accesses to the GIC registers' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

ICV_PMR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|----------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | Priority | | | | | | | |

Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The priority mask level for the virtual CPU interface. If the priority of a virtual interrupt is higher than the value indicated by this field, the interface signals the virtual interrupt to the PE.

The possible priority field values are as follows:

| Implemented priority bits | Possible priority field values | Number of priority levels |
|---------------------------|-------------------------------------|---------------------------|
| [7:0] | 0x00-0xFF (0-255), all values | 256 |
| [7:1] | 0x00-0xFE (0-254), even values only | 128 |
| [7:2] | 0x00-0xFC (0-252), in steps of 4 | 64 |
| [7:3] | 0x00-0xF8 (0-248), in steps of 8 | 32 |
| [7:4] | 0x00-0xF0 (0-240), in steps of 16 | 16 |

Unimplemented priority bits are RAZ/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing ICV_PMR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0100 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_PMR;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_PMR;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_PMR;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_PMR;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_PMR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        return ICC_PMR;
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        return ICC_PMR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b0100 | 0b0110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        ICV_PMR = R[t];
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        ICV_PMR = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        ICV_PMR = R[t];
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        ICV_PMR = R[t];
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_PMR = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif ICC_HSRE.SRE == '0' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch32.TakeMonitorTrapException();
    else
        ICC_PMR = R[t];
elsif PSTATE.EL == EL3 then
    if ICC_MSRE.SRE == '0' then
        UNDEFINED;
    else
        ICC_PMR = R[t];

```


ICV_RPR, Interrupt Controller Virtual Running Priority Register

The ICV_RPR characteristics are:

Purpose

Indicates the Running priority of the virtual CPU interface.

Configuration

AArch32 System register ICV_RPR performs the same function as AArch64 System register [ICV_RPR_EL1](#).

This register is present only when AArch32 is supported, FEAT_GICv3 is implemented and EL2 is implemented. Otherwise, direct accesses to ICV_RPR are UNDEFINED.

Attributes

ICV_RPR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|----------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | Priority | | | | | | | |

Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The current running priority on the virtual CPU interface. This is the group priority of the current active virtual interrupt.

The priority returned is the group priority as if the BPR for the current Exception level and Security state was set to the minimum value of BPR for the number of implemented priority bits.

Note

If 8 bits of priority are implemented the group priority is bits[7:1] of the priority.

Accessing ICV_RPR

If there are no active interrupts on the virtual CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

Software cannot determine the number of implemented priority bits from a read of this register.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| | | | | |
|--------|------|-----|-----|------|
| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b1011 | 0b011 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> ==
'11' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && ICH_HCR_EL2.TC == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ICH_HCR.TC == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FMO == '1' then
        return ICV_RPR;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.IMO == '1' then
        return ICV_RPR;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FMO == '1' then
        return ICV_RPR;
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.IMO == '1' then
        return ICV_RPR;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SCR.<IRQ,FIQ> == '11'
then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_RPR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
            UNDEFINED;
        elsif ICC_HSRE.SRE == '0' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && SCR_EL3.<IRQ,FIQ> == '11' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SCR.<IRQ,FIQ> == '11' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return ICC_RPR;
    elsif PSTATE.EL == EL3 then
        if ICC_MSRE.SRE == '0' then
            UNDEFINED;
        else
            return ICC_RPR;

```


ID_AFR0, Auxiliary Feature Register 0

The ID_AFR0 characteristics are:

Purpose

Provides information about the IMPLEMENTATION DEFINED features of the PE in AArch32 state.

Must be interpreted with the Main ID Register, [MIDR](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_AFR0 bits [31:0] are architecturally mapped to AArch64 System register [ID_AFR0_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_AFR0 are UNDEFINED.

Attributes

ID_AFR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------------|----|----|----|------------------------|----|---|---|------------------------|---|---|---|------------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | IMPLEMENTATION DEFINED | | | | IMPLEMENTATION DEFINED | | | | IMPLEMENTATION DEFINED | | | | IMPLEMENTATION DEFINED | | | |

Bits [31:16]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [15:12]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [11:8]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [7:4]

IMPLEMENTATION DEFINED.

IMPLEMENTATION DEFINED, bits [3:0]

IMPLEMENTATION DEFINED.

Accessing ID_AFR0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_AFR0;
elseif PSTATE.EL == EL2 then
    return ID_AFR0;
elseif PSTATE.EL == EL3 then
    return ID_AFR0;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_DFR0, Debug Feature Register 0

The ID_DFR0 characteristics are:

Purpose

Provides top level information about the debug system in AArch32 state.

Must be interpreted with the Main ID Register, [MIDR](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_DFR0 bits [31:0] are architecturally mapped to AArch64 System register [ID_DFR0_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_DFR0 are UNDEFINED.

Attributes

ID_DFR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------------|----|----|----|-------------------------|----|----|----|--------------------------|----|----|----|-------------------------|----|----|----|------------------------|----|----|----|-------------------------|----|---|---|-------------------------|---|---|---|------------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TraceFilt | | | | PerfMon | | | | MProfDbg | | | | MMapTrc | | | | CopTrc | | | | MMapDbg | | | | CopSDBG | | | | CopDbg | | | |

TraceFilt, bits [31:28]

Armv8.4 Self-hosted Trace Extension version. Defined values are:

| TraceFilt | Meaning |
|-----------|--|
| 0b0000 | Armv8.4 Self-hosted Trace Extension not implemented. |
| 0b0001 | Armv8.4 Self-hosted Trace Extension implemented. |

All other values are reserved.

FEAT_TRF implements the functionality added by the value 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

PerfMon, bits [27:24]

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version'.

Defined values are:

| PerfMon | Meaning |
|---------|---|
| 0b0000 | Performance Monitors Extension not implemented. |
| 0b0001 | Performance Monitors Extension, PMUv1 implemented. |
| 0b0010 | Performance Monitors Extension, PMUv2 implemented. |
| 0b0011 | Performance Monitors Extension, PMUv3 implemented. |
| 0b0100 | PMUv3 for Armv8.1. As 0b0011, and adds support for: <ul style="list-style-type: none"> Extended 16-bit PMEVTYPER<n>.evtCount field. If EL2 is implemented, the HDCR.HPMD control. |
| 0b0101 | PMUv3 for Armv8.4. As 0b0100, and adds support for the PMMIR register. |
| 0b0110 | PMUv3 for Armv8.5. As 0b0101, and adds support for: <ul style="list-style-type: none"> 64-bit event counters. If EL2 is implemented, the HDCR.HCCD control. If EL3 is implemented, the SDCR.SCCD control. |
| 0b0111 | PMUv3 for Armv8.7. As 0b0110, and adds support for: <ul style="list-style-type: none"> The PMCR.FZO and, if EL2 is implemented, HDCR.HPMFZO controls. If EL3 is implemented and using AArch64, the MDCR_EL3.{MPMX,MCCD} controls. |
| 0b1000 | PMUv3 for Armv8.8. As 0b0111, and: <ul style="list-style-type: none"> Extends the Common event number space to include 0x0040 to 0x00BF and 0x4040 to 0x40BF. Removes the CONSTRAINED UNPREDICTABLE behaviors if a reserved or unimplemented PMU event number is selected. |
| 0b1111 | IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value for new implementations. |

All other values are reserved.

FEAT_PMUv3 implements the functionality identified by the value 0b0011.

FEAT_PMUv3p1 implements the functionality identified by the value 0b0100.

FEAT_PMUv3p4 implements the functionality identified by the value 0b0101.

FEAT_PMUv3p5 implements the functionality identified by the value 0b0110.

FEAT_PMUv3p7 implements the functionality identified by the value 0b0111.

FEAT_PMUv3p8 implements the functionality identified by the value 0b1000.

In any Armv8 implementation, the values 0b0001 and 0b0010 are not permitted.

From Armv8.1, if FEAT_PMUv3 is implemented, the value 0b0011 is not permitted.

From Armv8.4, if FEAT_PMUv3 is implemented, the value 0b0100 is not permitted.

From Armv8.5, if FEAT_PMUv3 is implemented, the value 0b0101 is not permitted.

From Armv8.7, if FEAT_PMUv3 is implemented, the value 0b0110 is not permitted.

From Armv8.8, if FEAT_PMUv3 is implemented, the value 0b0111 is not permitted.

Note

In Armv7, the value 0b0000 can mean that PMUv1 is implemented. PMUv1 is not permitted in an Armv8 implementation.

MProfDbg, bits [23:20]

M-profile Debug. Support for memory-mapped debug model for M-profile processors. Defined values are:

| MProfDbg | Meaning |
|----------|--|
| 0b0000 | Not supported. |
| 0b0001 | Support for M-profile Debug architecture, with memory-mapped access. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

MMapTrc, bits [19:16]

Memory-mapped Trace. Support for memory-mapped trace model. Defined values are:

| MMapTrc | Meaning |
|---------|--|
| 0b0000 | Not supported. |
| 0b0001 | Support for Arm trace architecture, with memory-mapped access. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

For more information, see the ARM® Embedded Trace Macrocell Architecture Specification, ETMv4 (ARM IHI 0064).

CopTrc, bits [15:12]

Support for System registers-based trace model, using registers in the coproc == 0b1110 encoding space. Defined values are:

| CopTrc | Meaning |
|--------|---|
| 0b0000 | Not supported. |
| 0b0001 | Support for Arm trace architecture, with System registers access. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

For more information, see the ARM® Embedded Trace Macrocell Architecture Specification, ETMv4 (ARM IHI 0064).

MMapDbg, bits [11:8]

Memory-mapped Debug. Support for Armv7 memory-mapped debug model for A and R-profile processors. Defined values are:

| MMapDbg | Meaning |
|---------|--|
| 0b0000 | Not supported. |
| 0b0100 | Support for Armv7, v7 Debug architecture, with memory-mapped access. |
| 0b0101 | Support for Armv7, v7.1 Debug architecture, with memory-mapped access. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

The optional memory map defined by Armv8 is not compatible with Armv7.

CopSDBG, bits [7:4]

Support for a System registers-based Secure debug model, using registers in the coproc = 0b1110 encoding space, for an A-profile processor that includes EL3.

If EL3 is not implemented and the implemented Security state is Non-secure state, this field is RES0. Otherwise, this field reads the same as bits [3:0].

CopDbg, bits [3:0]

Debug architecture version. Indicates presence of Armv8 debug architecture. Defined values are:

| CopDbg | Meaning |
|--------|---|
| 0b0000 | Not supported. |
| 0b0010 | Armv6, v6 Debug architecture, with System registers access. |
| 0b0011 | Armv6, v6.1 Debug architecture, with System registers access. |
| 0b0100 | Armv7, v7 Debug architecture, with System registers access. |
| 0b0101 | Armv7, v7.1 Debug architecture, with System registers access. |
| 0b0110 | Armv8 debug architecture. |
| 0b0111 | Armv8 debug architecture with Virtualization Host Extensions. |
| 0b1000 | Armv8.2 debug architecture, FEAT_Debugv8p2. |
| 0b1001 | Armv8.4 debug architecture, FEAT_Debugv8p4. |
| 0b1010 | Armv8.8 debug architecture, FEAT_Debugv8p8. |

All other values are reserved.

The values 0b0000, 0b0010, 0b0011, 0b0100, and 0b0101 are not permitted in Armv8.

FEAT_VHE adds the functionality identified by the value 0b0111.

FEAT_Debugv8p2 adds the functionality identified by the value 0b1000.

FEAT_Debugv8p4 adds the functionality identified by the value 0b1001.

FEAT_Debugv8p8 adds the functionality identified by the value 0b1010.

From Armv8.1, when FEAT_VHE is implemented the value 0b0110 is not permitted.

From Armv8.2, the values 0b0110 and 0b0111 are not permitted.

From Armv8.4, the value 0b1000 is not permitted.

From Armv8.8, the value 0b1001 is not permitted.

Accessing ID_DFR0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_DFR0;
elsif PSTATE.EL == EL2 then
    return ID_DFR0;
elsif PSTATE.EL == EL3 then
    return ID_DFR0;

```


ID_DFR1, Debug Feature Register 1

The ID_DFR1 characteristics are:

Purpose

Provides top level information about the debug system in AArch32.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_DFR1 bits [31:0] are architecturally mapped to AArch64 System register [ID_DFR1_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_DFR1 are UNDEFINED.

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_DFR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---|-------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | HPMNO | | | | MTPMU | | | |

Bits [31:8]

Reserved, RES0.

HPMNO, bits [7:4]

Zero PMU event counters for a Guest operating system. Defined values are:

| HPMNO | Meaning |
|--------|--|
| 0b0000 | Setting HDCR .HPMN to zero has CONSTRAINED UNPREDICTABLE behavior. |
| 0b0001 | Setting HDCR .HPMN to zero has defined behavior. |

All other values are reserved.

If FEAT_PMUv3 is not implemented, FEAT_FGT is not implemented, or EL2 is not implemented, the only permitted value is 0b0000.

FEAT_HPMNO implements the functionality identified by the value 0b0001.

From Armv8.8, in an implementation that includes FEAT_PMUv3, FEAT_FGT, and EL2, the value 0b0000 is not permitted.

MTPMU, bits [3:0]

Multi-threaded PMU extension. Defined values are:

| MTPMU | Meaning |
|--------|--|
| 0b0000 | FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, it is IMPLEMENTATION DEFINED whether PMEVTYPER<n>. are read/write or RES0. |
| 0b0001 | FEAT_MTPMU and FEAT_PMUv3 implemented. PMEVTYPER<n>. are read/write. When FEAT_MTPMU is disabled, the Effective values of PMEVTYPER<n>. are 0. |
| 0b1111 | FEAT_MTPMU not implemented. If FEAT_PMUv3 is implemented, PMEVTYPER<n>. are RES0. |

All other values are reserved.

FEAT_MTPMU implements the functionality identified by the value 0b0001.

From Armv8.6, in an implementation that includes FEAT_PMUv3, the value 0b0000 is not permitted.

In an implementation that does not include FEAT_PMUv3, the value 0b0001 is not permitted.

Accessing ID_DFR1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0011 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_DFR1) || boolean
IMPLEMENTATION_DEFINED "ID_DFR1 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && (!IsZero(ID_DFR1) || boolean
IMPLEMENTATION_DEFINED "ID_DFR1 trapped by HCR.TID3") && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_DFR1;
elsif PSTATE.EL == EL2 then
    return ID_DFR1;
elsif PSTATE.EL == EL3 then
    return ID_DFR1;

```

ID_ISAR0, Instruction Set Attribute Register 0

The ID_ISAR0 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR1](#), [ID_ISAR2](#), [ID_ISAR3](#), [ID_ISAR4](#), and [ID_ISAR5](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_ISAR0 bits [31:0] are architecturally mapped to AArch64 System register [ID_ISAR0_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_ISAR0 are UNDEFINED.

Attributes

ID_ISAR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|--------|----|----|----|-------|----|----|----|---------|----|----|----|-----------|----|----|----|----------|----|---|---|----------|---|---|---|------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | Divide | | | | Debug | | | | Coprocc | | | | CmpBranch | | | | BitField | | | | BitCount | | | | Swap | | | |

Bits [31:28]

Reserved, RES0.

Divide, bits [27:24]

Indicates the implemented Divide instructions. Defined values are:

| Divide | Meaning |
|--------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds SDIV and UDIV in the T32 instruction set. |
| 0b0010 | As for 0b0001, and adds SDIV and UDIV in the A32 instruction set. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Debug, bits [23:20]

Indicates the implemented Debug instructions. Defined values are:

| Debug | Meaning |
|--------|-------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds BKPT. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Coproc, bits [19:16]

Indicates the implemented System register access instructions. Defined values are:

| Coproc | Meaning |
|---------------|--|
| 0b0000 | None implemented, except for instructions separately attributed by the architecture to provide access to AArch32 System registers and System instructions. |
| 0b0001 | Adds generic CDP, LDC, MCR, MRC, and STC. |
| 0b0010 | As for 0b0001, and adds generic CDP2, LDC2, MCR2, MRC2, and STC2. |
| 0b0011 | As for 0b0010, and adds generic MCRR and MRRC. |
| 0b0100 | As for 0b0011, and adds generic MCRR2 and MRRC2. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

CmpBranch, bits [15:12]

Indicates the implemented combined Compare and Branch instructions in the T32 instruction set. Defined values are:

| CmpBranch | Meaning |
|------------------|--------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds CBNZ and CBZ. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

BitField, bits [11:8]

Indicates the implemented BitField instructions. Defined values are:

| BitField | Meaning |
|-----------------|--------------------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds BFC, BFI, SBFX, and UBFX. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

BitCount, bits [7:4]

Indicates the implemented Bit Counting instructions. Defined values are:

| BitCount | Meaning |
|-----------------|-------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds CLZ. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Swap, bits [3:0]

Indicates the implemented Swap instructions in the A32 instruction set. Defined values are:

| Swap | Meaning |
|-------------|--------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds SWP and SWPB. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Accessing ID_ISAR0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_ISAR0;
elsif PSTATE.EL == EL2 then
    return ID_ISAR0;
elsif PSTATE.EL == EL3 then
    return ID_ISAR0;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR1, Instruction Set Attribute Register 1

The ID_ISAR1 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0](#), [ID_ISAR2](#), [ID_ISAR3](#), [ID_ISAR4](#), and [ID_ISAR5](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_ISAR1 bits [31:0] are architecturally mapped to AArch64 System register [ID_ISAR1_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_ISAR1 are UNDEFINED.

Attributes

ID_ISAR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------|---------------------------|---------------------------|------------------------|------------------------|---------------------------|------------------------|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Jazelle | Interwork | Immediate | IfThen | Extend | Except_AR | Except | Endian | | | | | | | | | | | | | | | | | | | | | | | | |

Jazelle, bits [31:28]

Indicates the implemented Jazelle extension instructions. Defined values are:

| Jazelle | Meaning |
|---------|--|
| 0b0000 | No support for Jazelle. |
| 0b0001 | Adds the BXJ instruction, and the J bit in the PSR. This setting might indicate a trivial implementation of the Jazelle extension. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Interwork, bits [27:24]

Indicates the implemented Interworking instructions. Defined values are:

| Interwork | Meaning |
|-----------|--|
| 0b0000 | None implemented. |
| 0b0001 | Adds the BX instruction, and the T bit in the PSR. |
| 0b0010 | As for 0b0001, and adds the BLX instruction. PC loads have BX-like behavior. |
| 0b0011 | As for 0b0010, and guarantees that data-processing instructions in the A32 instruction set with the PC as the destination and the S bit clear have BX-like behavior. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

Immediate, bits [23:20]

Indicates the implemented data-processing instructions with long immediates. Defined values are:

| Immediate | Meaning |
|-----------|--|
| 0b0000 | None implemented. |
| 0b0001 | Adds: <ul style="list-style-type: none"> • The MOVT instruction • The MOV instruction encodings with zero-extended 16-bit immediates. • The T32 ADD and SUB instruction encodings with zero-extended 12-bit immediates, and the other ADD, ADR, and SUB encodings cross-referenced by the pseudocode for those encodings. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

IfThen, bits [19:16]

Indicates the implemented If-Then instructions in the T32 instruction set. Defined values are:

| IfThen | Meaning |
|--------|--|
| 0b0000 | None implemented. |
| 0b0001 | Adds the IT instructions, and the IT bits in the PSRs. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Extend, bits [15:12]

Indicates the implemented Extend instructions. Defined values are:

| Extend | Meaning |
|--------|--|
| 0b0000 | No scalar sign-extend or zero-extend instructions are implemented, where scalar instructions means non-Advanced SIMD instructions. |
| 0b0001 | Adds the SXTB, SXTB, UXTB, and UXTB instructions. |
| 0b0010 | As for 0b0001, and adds the SXTB16, SXTAB, SXTAB16, SXTAH, UXTB16, UXTAB, UXTAB16, and UXTAH instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Except_AR, bits [11:8]

Indicates the implemented A and R-profile exception-handling instructions. Defined values are:

| Except_AR | Meaning |
|-----------|--|
| 0b0000 | None implemented. |
| 0b0001 | Adds the SRS and RFE instructions, and the A and R-profile forms of the CPS instruction. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Except, bits [7:4]

Indicates the implemented exception-handling instructions in the A32 instruction set. Defined values are:

| Except | Meaning |
|--------|--|
| 0b0000 | Not implemented. This indicates that the User bank and Exception return forms of the LDM and STM instructions are not implemented. |
| 0b0001 | Adds the LDM (exception return), LDM (user registers), and STM (user registers) instruction versions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Endian, bits [3:0]

Indicates the implemented Endian instructions. Defined values are:

| Endian | Meaning |
|--------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds the SETEND instruction, and the E bit in the PSRs. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Accessing ID_ISAR1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_ISAR1;
elseif PSTATE.EL == EL2 then
    return ID_ISAR1;
elseif PSTATE.EL == EL3 then
    return ID_ISAR1;

```

ID_ISAR2, Instruction Set Attribute Register 2

The ID_ISAR2 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0](#), [ID_ISAR1](#), [ID_ISAR3](#), [ID_ISAR4](#), and [ID_ISAR5](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_ISAR2 bits [31:0] are architecturally mapped to AArch64 System register [ID_ISAR2_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_ISAR2 are UNDEFINED.

Attributes

ID_ISAR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|--------|-------|-------|------|----------------|---------|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Reversal | PSR_AR | MultU | MultS | Mult | MultiAccessInt | MemHint | LoadStore | | | | | | | | | | | | | | | | | | | | | | | | |

Reversal, bits [31:28]

Indicates the implemented Reversal instructions. Defined values are:

| Reversal | Meaning |
|----------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds the REV, REV16, and REVSH instructions. |
| 0b0010 | As for 0b0001, and adds the RBIT instruction. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

PSR_AR, bits [27:24]

Indicates the implemented A and R-profile instructions to manipulate the PSR. Defined values are:

| PSR_AR | Meaning |
|--------|--|
| 0b0000 | None implemented. |
| 0b0001 | Adds the MRS and MSR instructions, and the exception return forms of data-processing instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

The exception return forms of the data-processing instructions are:

- In the A32 instruction set, data-processing instructions with the PC as the destination and the S bit set. These instructions might be affected by the WithShifts attribute.
- In the T32 instruction set, the SUBS PC,LR,#N instruction.

MultU, bits [23:20]

Indicates the implemented advanced unsigned Multiply instructions. Defined values are:

| MultU | Meaning |
|--------------|--|
| 0b0000 | None implemented. |
| 0b0001 | Adds the UMULL and UMLAL instructions. |
| 0b0010 | As for 0b0001, and adds the UMAAL instruction. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

MultS, bits [19:16]

Indicates the implemented advanced signed Multiply instructions. Defined values are:

| MultS | Meaning |
|--------------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds the SMULL and SMLAL instructions. |
| 0b0010 | As for 0b0001, and adds the SMLABB, SMLABT, SMLALBB, SMLALBT, SMLALTB, SMLALTT, SMLATB, SMLATT, SMLAWB, SMLAWT, SMULBB, SMULBT, SMULTB, SMULTT, SMULWB, and SMULWT instructions. Also adds the Q bit in the PSRs. |
| 0b0011 | As for 0b0010, and adds the SMLAD, SMLADX, SMLALD, SMLALDX, SMLSD, SMLSDX, SMLSXD, SMLSXD, SMMLA, SMMLAR, SMMLS, SMMLSR, SMMUL, SMMULR, SMUAD, SMUADX, SMUSD, and SMUSDX instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

Mult, bits [15:12]

Indicates the implemented additional Multiply instructions. Defined values are:

| Mult | Meaning |
|-------------|---|
| 0b0000 | No additional instructions implemented. This means only MUL is implemented. |
| 0b0001 | Adds the MLA instruction. |
| 0b0010 | As for 0b0001, and adds the MLS instruction. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

MultiAccessInt, bits [11:8]

Indicates the support for interruptible multi-access instructions. Defined values are:

| MultiAccessInt | Meaning |
|-----------------------|--|
| 0b0000 | No support. This means the LDM and STM instructions are not interruptible. |
| 0b0001 | LDM and STM instructions are restartable. |
| 0b0010 | LDM and STM instructions are continuable. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

MemHint, bits [7:4]

Indicates the implemented Memory Hint instructions. Defined values are:

| MemHint | Meaning |
|---------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds the PLD instruction. |
| 0b0010 | Adds the PLD instruction. (0b0001 and 0b0010 have identical effects.) |
| 0b0011 | As for 0b0001 (or 0b0010), and adds the PLI instruction. |
| 0b0100 | As for 0b0011, and adds the PLDW instruction. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0100.

LoadStore, bits [3:0]

Indicates the implemented additional load/store instructions. Defined values are:

| LoadStore | Meaning |
|-----------|--|
| 0b0000 | No additional load/store instructions implemented. |
| 0b0001 | Adds the LDRD and STRD instructions. |
| 0b0010 | As for 0b0001, and adds the Load Acquire (LDAB, LDAH, LDA, LDAEXB, LDAEXH, LDAEX, LDAEXD) and Store Release (STLB, STLH, STL, STLEXB, STLEXH, STLEX, STLEXD) instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Accessing ID_ISAR2

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_ISAR2;
elsif PSTATE.EL == EL2 then
    return ID_ISAR2;
elsif PSTATE.EL == EL3 then
    return ID_ISAR2;

```

ID_ISAR3, Instruction Set Attribute Register 3

The ID_ISAR3 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0](#), [ID_ISAR1](#), [ID_ISAR2](#), [ID_ISAR4](#), and [ID_ISAR5](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_ISAR3 bits [31:0] are architecturally mapped to AArch64 System register [ID_ISAR3_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_ISAR3 are UNDEFINED.

Attributes

ID_ISAR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------|-------------------------|-------------------------|---------------------------|---------------------------|---------------------|----------------------|--------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| T32EE | TrueNOP | T32Copy | TabBranch | SynchPrim | SVC | SIMD | Saturate | | | | | | | | | | | | | | | | | | | | | | | | |

T32EE, bits [31:28]

Indicates the implemented T32EE instructions. Defined values are:

| T32EE | Meaning |
|--------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds the ENTERX and LEAVEX instructions, and modifies the load behavior to include null checking. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

TrueNOP, bits [27:24]

Indicates the implemented true NOP instructions. Defined values are:

| TrueNOP | Meaning |
|---------|---|
| 0b0000 | None implemented. This means there are no NOP instructions that do not have any register dependencies. |
| 0b0001 | Adds true NOP instructions in both the T32 and A32 instruction sets. This also permits additional NOP-compatible hints. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

T32Copy, bits [23:20]

Indicates the support for T32 non flag-setting MOV instructions. Defined values are:

| T32Copy | Meaning |
|----------------|---|
| 0b0000 | Not supported. This means that in the T32 instruction set, encoding T1 of the MOV (register) instruction does not support a copy from a low register to a low register. |
| 0b0001 | Adds support for T32 instruction set encoding T1 of the MOV (register) instruction, copying from a low register to a low register. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

TabBranch, bits [19:16]

Indicates the implemented Table Branch instructions in the T32 instruction set. Defined values are:

| TabBranch | Meaning |
|------------------|------------------------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds the TBB and TBH instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

SynchPrim, bits [15:12]

Used in conjunction with ID_ISAR4.SynchPrim_frac to indicate the implemented Synchronization Primitive instructions. Defined values are:

| SynchPrim | Meaning |
|------------------|---|
| 0b0000 | If SynchPrim_frac == 0b000, no Synchronization Primitives implemented. |
| 0b0001 | If SynchPrim_frac == 0b000, adds the LDREX and STREX instructions. |
| 0b0010 | If SynchPrim_frac == 0b011, also adds the CLREX, LDREXB, STREXB, and STREXH instructions. |
| 0b0011 | If SynchPrim_frac == 0b000, as for [0b001, 0b011] and also adds the LDREXD and STREXD instructions. |

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A, the only permitted value is 0b0010.

SVC, bits [11:8]

Indicates the implemented SVC instructions. Defined values are:

| SVC | Meaning |
|------------|---------------------------|
| 0b0000 | Not implemented. |
| 0b0001 | Adds the SVC instruction. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

SIMD, bits [7:4]

Indicates the implemented SIMD instructions. Defined values are:

| SIMD | Meaning |
|--------|--|
| 0b0000 | None implemented. |
| 0b0001 | Adds the SSAT and USAT instructions, and the Q bit in the PSRs. |
| 0b0011 | As for 0b0001, and adds the PKHBT, PKHTB, QADD16, QADD8, QASX, QSUB16, QSUB8, QSAX, SADD16, SADD8, SASX, SEL, SHADD16, SHADD8, SHASX, SHSUB16, SHSUB8, SHSAX, SSAT16, SSUB16, SSUB8, SSAX, SXTAB16, SXTB16, UADD16, UADD8, UASX, UHADD16, UHADD8, UHASX, UHSUB16, UHSUB8, UHSAX, UQADD16, UQADD8, UQASX, UQSUB16, UQSUB8, UQSAX, USAD8, USADA8, USAT16, USUB16, USUB8, USAX, UXTAB16, and UXTB16 instructions. Also adds support for the GE[3:0] bits in the PSRs. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

The SIMD field relates only to implemented instructions that perform SIMD operations on the general-purpose registers. In an implementation that supports Advanced SIMD and floating-point instructions, [MVFR0](#), [MVFR1](#), and [MVFR2](#) give information about the implemented Advanced SIMD instructions.

Saturate, bits [3:0]

Indicates the implemented Saturate instructions. Defined values are:

| Saturate | Meaning |
|----------|--|
| 0b0000 | None implemented. This means no non-Advanced SIMD saturate instructions are implemented. |
| 0b0001 | Adds the QADD, QDADD, QDSUB, and QSUB instructions, and the Q bit in the PSRs. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Accessing ID_ISAR3

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_ISAR3;
elsif PSTATE.EL == EL2 then
    return ID_ISAR3;
elsif PSTATE.EL == EL3 then
    return ID_ISAR3;

```


ID_ISAR4, Instruction Set Attribute Register 4

The ID_ISAR4 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0](#), [ID_ISAR1](#), [ID_ISAR2](#), [ID_ISAR3](#), and [ID_ISAR5](#).

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_ISAR4 bits [31:0] are architecturally mapped to AArch64 System register [ID_ISAR4_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_ISAR4 are UNDEFINED.

Attributes

ID_ISAR4 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------------|----|----|----|-----------------------|----|----|----|--------------------------------|----|----|----|-------------------------|----|----|----|---------------------|----|----|----|---------------------------|----|---|---|----------------------------|---|---|---|------------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SWP_frac | | | | PSR_M | | | | SynchPrim_frac | | | | Barrier | | | | SMC | | | | Writeback | | | | WithShifts | | | | Unpriv | | | |

SWP_frac, bits [31:28]

Indicates support for the memory system locking the bus for SWP or SWPB instructions. Defined values are:

| SWP_frac | Meaning |
|----------|---|
| 0b0000 | SWP or SWPB instructions not implemented. |
| 0b0001 | SWP or SWPB implemented but only in a uniprocessor context. SWP and SWPB do not guarantee whether memory accesses from other Requesters can come between the load memory access and the store memory access of the SWP or SWPB. |

All other values are reserved. This field is valid only if [ID_ISAR0](#).Swap is 0b0000.

In Armv8-A, the only permitted value is 0b0000.

PSR_M, bits [27:24]

Indicates the implemented M-profile instructions to modify the PSRs. Defined values are:

| PSR_M | Meaning |
|--------|---|
| 0b0000 | None implemented. |
| 0b0001 | Adds the M-profile forms of the CPS, MRS, and MSR instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

SynchPrim_frac, bits [23:20]

Used in conjunction with [ID_ISAR3.SynchPrim](#) to indicate the implemented Synchronization Primitive instructions. Possible values are:

| SynchPrim_frac | Meaning |
|-----------------------|---|
| 0b0000 | If SynchPrim == 0b0000, no Synchronization Primitives implemented. If SynchPrim == 0b0001, adds the LDREX and STREX instructions. If SynchPrim == 0b0010, also adds the CLREX, LDREXB, LDREXH, STREXB, STREXH, LDREXD, and STREXD instructions. |
| 0b0011 | If SynchPrim == 0b0001, adds the LDREX, STREX, CLREX, LDREXB, LDREXH, STREXB, and STREXH instructions. |

All other combinations of SynchPrim and SynchPrim_frac are reserved.

In Armv8-A, the only permitted value is 0b0000.

Barrier, bits [19:16]

Indicates the implemented Barrier instructions in the A32 and T32 instruction sets. Defined values are:

| Barrier | Meaning |
|----------------|---|
| 0b0000 | None implemented. Barrier operations are provided only as System instructions in the (coproc==0b1111) encoding space. |
| 0b0001 | Adds the DMB, DSB, and ISB barrier instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

SMC, bits [15:12]

Indicates the implemented SMC instructions. Defined values are:

| SMC | Meaning |
|------------|---------------------------|
| 0b0000 | None implemented. |
| 0b0001 | Adds the SMC instruction. |

All other values are reserved.

In Armv8-A, the permitted values are:

- If EL3 is implemented, the only permitted value is 0b0001.
- If neither EL3 nor EL2 is implemented, the only permitted value is 0b0000.

Writeback, bits [11:8]

Indicates the support for Writeback addressing modes. Defined values are:

| Writeback | Meaning |
|------------------|--|
| 0b0000 | Basic support. Only the LDM, STM, PUSH, POP, SRS, and RFE instructions support writeback addressing modes. These instructions support all of their writeback addressing modes. |
| 0b0001 | Adds support for all of the writeback addressing modes. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

WithShifts, bits [7:4]

Indicates the support for instructions with shifts. Defined values are:

| WithShifts | Meaning |
|-------------------|--|
| 0b0000 | Nonzero shifts supported only in MOV and shift instructions. |
| 0b0001 | Adds support for shifts of loads and stores over the range LSL 0-3. |
| 0b0011 | As for 0b0001, and adds support for other constant shift options, both on load/store and other instructions. |
| 0b0100 | As for 0b0011, and adds support for register-controlled shift options. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0100.

Unpriv, bits [3:0]

Indicates the implemented unprivileged instructions. Defined values are:

| Unpriv | Meaning |
|---------------|--|
| 0b0000 | None implemented. No T variant instructions are implemented. |
| 0b0001 | Adds the LDRBT, LDRT, STRBT, and STRT instructions. |
| 0b0010 | As for 0b0001, and adds the LDRHT, LDRSBT, LDRSHT, and STRHT instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Accessing ID_ISAR4

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_ISAR4;
elsif PSTATE.EL == EL2 then
    return ID_ISAR4;
elsif PSTATE.EL == EL3 then
    return ID_ISAR4;

```


ID_ISAR5, Instruction Set Attribute Register 5

The ID_ISAR5 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0](#), [ID_ISAR1](#), [ID_ISAR2](#), [ID_ISAR3](#), and [ID_ISAR4](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_ISAR5 bits [31:0] are architecturally mapped to AArch64 System register [ID_ISAR5_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_ISAR5 are UNDEFINED.

Attributes

ID_ISAR5 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-----|----|----|----|------|----|----|----|-------|----|----|----|------|----|----|----|------|----|---|---|-----|---|---|---|------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VCMA | | | | RDM | | | | RES0 | | | | CRC32 | | | | SHA2 | | | | SHA1 | | | | AES | | | | SEVL | | | |

VCMA, bits [31:28]

Indicates AArch32 support for complex number addition and multiplication where numbers are stored in vectors. Defined values are:

| VCMA | Meaning |
|--------|--|
| 0b0000 | The VCMLA and VCADD instructions are not implemented in AArch32. |
| 0b0001 | The VCMLA and VCADD instructions are implemented in AArch32. |

All other values are reserved.

FEAT_FCMA implements the functionality identified by 0b0001.

From Armv8.3, the only permitted value is 0b0001.

RDM, bits [27:24]

Indicates support for the VQRDMLAH and VQRDMLSH instructions in AArch32 state. Defined values are:

| RDM | Meaning |
|--------|--|
| 0b0000 | No VQRDMLAH and VQRDMLSH instructions implemented. |
| 0b0001 | VQRDMLAH and VQRDMLSH instructions implemented. |

All other values are reserved.

FEAT_RDM implements the functionality identified by the value 0b0001.

From Armv8.1, the only permitted value is 0b0001.

Bits [23:20]

Reserved, RES0.

CRC32, bits [19:16]

Indicates support for the CRC32 instructions in AArch32 state. Defined values are:

| CRC32 | Meaning |
|--------------|---|
| 0b0000 | No CRC32 instructions implemented. |
| 0b0001 | CRC32B, CRC32H, CRC32W, CRC32CB, CRC32CH, and CRC32CW instructions implemented. |

All other values are reserved.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.1, the only permitted value is 0b0001.

SHA2, bits [15:12]

Indicates support for the SHA2 instructions in AArch32 state.

| SHA2 | Meaning |
|-------------|--|
| 0b0000 | No SHA2 instructions implemented. |
| 0b0001 | SHA256H, SHA256H2, SHA256SU0, and SHA256SU1 implemented. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

SHA1, bits [11:8]

Indicates support for the SHA1 instructions are implemented in AArch32 state. Defined values are:

| SHA1 | Meaning |
|-------------|---|
| 0b0000 | No SHA1 instructions implemented. |
| 0b0001 | SHA1C, SHA1P, SHA1M, SHA1H, SHA1SU0, and SHA1SU1 implemented. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

AES, bits [7:4]

Indicates support for the AES instructions in AArch32 state. Defined values are:

| AES | Meaning |
|------------|--|
| 0b0000 | No AES instructions implemented. |
| 0b0001 | AESE, AESD, AESMC, and AESIMC implemented. |
| 0b0010 | As for 0b0001, plus VMULL (polynomial) instructions operating on 64-bit data quantities. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0010.

SEVL, bits [3:0]

Indicates support for the SEVL instruction in AArch32 state. Defined values are:

| SEVL | Meaning |
|--------|--|
| 0b0000 | SEVL is implemented as a NOP. |
| 0b0001 | SEVL is implemented as Send Event Local. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Accessing ID_ISAR5

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_ISAR5;
elsif PSTATE.EL == EL2 then
    return ID_ISAR5;
elsif PSTATE.EL == EL3 then
    return ID_ISAR5;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_ISAR6, Instruction Set Attribute Register 6

The ID_ISAR6 characteristics are:

Purpose

Provides information about the instruction sets implemented by the PE in AArch32 state.

Must be interpreted with [ID_ISAR0](#), [ID_ISAR1](#), [ID_ISAR2](#), [ID_ISAR3](#), [ID_ISAR4](#), and [ID_ISAR5](#).

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_ISAR6 bits [31:0] are architecturally mapped to AArch64 System register [ID_ISAR6_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_ISAR6 are UNDEFINED.

Note

Prior to the introduction of the features described by this register, this register was unnamed and reserved, RES0 from EL1, EL2, and EL3.

Attributes

ID_ISAR6 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|----|----|----|----------------------|----|----|----|----------------------|----|----|----|-------------------------|----|----|----|--------------------|----|----|----|---------------------|----|---|---|--------------------|---|---|---|-----------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | I8MM | | | | BF16 | | | | SPECRES | | | | SB | | | | FHM | | | | DP | | | | JSCVT | | | |

Bits [31:28]

Reserved, RES0.

I8MM, bits [27:24]

Indicates support for Advanced SIMD and floating-point Int8 matrix multiplication instructions in AArch32 state. Defined values are:

| I8MM | Meaning |
|-------------|---|
| 0b0000 | Int8 matrix multiplication instructions are not implemented. |
| 0b0001 | VSMMLA, VSUDOT, VUMMLA, VUSMMLA, and VUSDOT instructions are implemented. |

All other values are reserved.

FEAT_AA32I8MM implements the functionality identified by 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

BF16, bits [23:20]

Indicates support for Advanced SIMD and floating-point BFloat16 instructions in AArch32 state. Defined values are:

| BF16 | Meaning |
|-------------|---|
| 0b0000 | BFloat16 instructions are not implemented. |
| 0b0001 | VCVT, VCVTB, VCVTT, VDOT, VFMA, VFMA, and VMMLA instructions with BF16 operand or result types are implemented. |

All other values are reserved.

FEAT_AA32BF16 implements the functionality identified by 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

SPECRES, bits [19:16]

Indicates support for Speculation invalidation instructions in AArch32 state. Defined values are:

| SPECRES | Meaning |
|----------------|---|
| 0b0000 | CFPRCTX, DVPRCTX, and CPPRCTX instructions are not implemented. |
| 0b0001 | CFPRCTX, DVPRCTX, and CPPRCTX instructions are implemented. |

All other values are reserved.

From Armv8.5, the only permitted value is 0b0001.

SB, bits [15:12]

Indicates support for SB instruction in AArch32 state. Defined values are:

| SB | Meaning |
|-----------|------------------------------------|
| 0b0000 | SB instruction is not implemented. |
| 0b0001 | SB instruction is implemented. |

All other values are reserved.

From Armv8.5, the only permitted value is 0b0001.

FHM, bits [11:8]

Indicates support for Advanced SIMD and floating-point VFMA and VFMSL instructions in AArch32 state. Defined values are:

| FHM | Meaning |
|------------|--|
| 0b0000 | VFMA and VFMSL instructions not implemented. |
| 0b0001 | VFMA and VFMSL instructions implemented. |

FEAT_FHM implements the functionality identified by the value 0b0001.

DP, bits [7:4]

Indicates support for dot product instructions in AArch32 state. Defined values are:

| DP | Meaning |
|-----------|---|
| 0b0000 | No dot product instructions implemented. |
| 0b0001 | VUDOT and VSDOT instructions implemented. |

All other values are reserved.

FEAT_DotProd implements the functionality identified by the value 0b0001.

JSCVT, bits [3:0]

Indicates support for the Javascript conversion instruction in AArch32 state. Defined values are:

| JSCVT | Meaning |
|--------------|---|
| 0b0000 | The VJCVT instruction is not implemented. |
| 0b0001 | The VJCVT instruction is implemented. |

All other values are reserved.

In Armv8.0, the only permitted value is 0b0000.

FEAT_JSCVT implements the functionality identified by 0b0001.

From Armv8.3, if Advanced SIMD or Floating-point is implemented, the only permitted value is 0b0001.

From Armv8.3, if Advanced SIMD or Floating-point is not implemented, the only permitted value is 0b0000.

Accessing ID_ISAR6

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_ISAR6) || boolean
IMPLEMENTATION_DEFINED "ID_ISAR6 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && (!IsZero(ID_ISAR6) || boolean
IMPLEMENTATION_DEFINED "ID_ISAR6 trapped by HCR.TID3") && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_ISAR6;
elseif PSTATE.EL == EL2 then
    return ID_ISAR6;
elseif PSTATE.EL == EL3 then
    return ID_ISAR6;

```

ID_MMFR0, Memory Model Feature Register 0

The ID_MMFR0 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_MMFR0 bits [31:0] are architecturally mapped to AArch64 System register [ID_MMFR0_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_MMFR0 are UNDEFINED.

Attributes

ID_MMFR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|------|----|----|----|--------|----|----|----|-----|----|----|----|----------|----|----|----|----------|----|---|---|------|---|---|---|------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| InnerShr | | | | FCSE | | | | AuxReg | | | | TCM | | | | ShareLvl | | | | OuterShr | | | | PMSA | | | | VMSA | | | |

InnerShr, bits [31:28]

Innermost Shareability. Indicates the innermost shareability domain implemented. Defined values are:

| InnerShr | Meaning |
|----------|--|
| 0b0000 | Implemented as Non-cacheable. |
| 0b0001 | Implemented with hardware coherency support. |
| 0b1111 | Shareability ignored. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000, 0b0001, and 0b1111.

This field is valid only if the implementation supports two levels of shareability, as indicated by ID_MMFR0.ShareLvl having the value 0b0001.

When ID_MMFR0.ShareLvl is zero, this field is UNKNOWN.

FCSE, bits [27:24]

Indicates whether the implementation includes the FCSE. Defined values are:

| FCSE | Meaning |
|--------|-------------------|
| 0b0000 | Not supported. |
| 0b0001 | Support for FCSE. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

AuxReg, bits [23:20]

Auxiliary Registers. Indicates support for Auxiliary registers. Defined values are:

| AuxReg | Meaning |
|---------------|--|
| 0b0000 | None supported. |
| 0b0001 | Support for Auxiliary Control Register only. |
| 0b0010 | Support for Auxiliary Fault Status Registers (AIFSR and ADFSR) and Auxiliary Control Register. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Note

Accesses to unimplemented Auxiliary registers are UNDEFINED.

TCM, bits [19:16]

Indicates support for TCMs and associated DMAs. Defined values are:

| TCM | Meaning |
|------------|---|
| 0b0000 | Not supported. |
| 0b0001 | Support is IMPLEMENTATION DEFINED. Armv7 requires this setting. |
| 0b0010 | Support for TCM only, Armv6 implementation. |
| 0b0011 | Support for TCM and DMA, Armv6 implementation. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

ShareLvl, bits [15:12]

Shareability Levels. Indicates the number of shareability levels implemented. Defined values are:

| ShareLvl | Meaning |
|-----------------|---|
| 0b0000 | One level of shareability implemented. |
| 0b0001 | Two levels of shareability implemented. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

OuterShr, bits [11:8]

Outermost Shareability. Indicates the outermost shareability domain implemented. Defined values are:

| OuterShr | Meaning |
|-----------------|--|
| 0b0000 | Implemented as Non-cacheable. |
| 0b0001 | Implemented with hardware coherency support. |
| 0b1111 | Shareability ignored. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000, 0b0001, and 0b1111.

PMSA, bits [7:4]

Indicates support for a PMSA. Defined values are:

| PMSA | Meaning |
|--------|---|
| 0b0000 | Not supported. |
| 0b0001 | Support for IMPLEMENTATION DEFINED PMSA. |
| 0b0010 | Support for PMSAv6, with a Cache Type Register implemented. |
| 0b0011 | Support for PMSAv7, with support for memory subsections. Armv7-R profile. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

VMSA, bits [3:0]

Indicates support for a VMSA. Defined values are:

| VMSA | Meaning |
|--------|---|
| 0b0000 | Not supported. |
| 0b0001 | Support for IMPLEMENTATION DEFINED VMSA. |
| 0b0010 | Support for VMSAv6, with Cache and TLB Type Registers implemented. |
| 0b0011 | Support for VMSAv7, with support for remapping and the Access flag. ARMv7-A profile. |
| 0b0100 | As for 0b0011, and adds support for the PXN bit in the Short-descriptor translation table format descriptors. |
| 0b0101 | As for 0b0100, and adds support for the Long-descriptor translation table format. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0101.

Accessing ID_MMFR0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_MMFR0;
elsif PSTATE.EL == EL2 then
    return ID_MMFR0;
elsif PSTATE.EL == EL3 then
    return ID_MMFR0;

```

ID_MMFR1, Memory Model Feature Register 1

The ID_MMFR1 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_MMFR1 bits [31:0] are architecturally mapped to AArch64 System register [ID_MMFR1_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_MMFR1 are UNDEFINED.

Attributes

ID_MMFR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------|----|----|----|--------------------------|----|----|----|-----------------------|----|----|----|-----------------------|----|----|----|-------------------------|----|----|----|-------------------------|----|---|---|-------------------------|---|---|---|-------------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| BPred | | | | L1TstCln | | | | L1Uni | | | | L1Hvd | | | | L1UniSW | | | | L1HvdSW | | | | L1UniVA | | | | L1HvdVA | | | |

BPred, bits [31:28]

Branch Predictor. Indicates branch predictor management requirements. Defined values are:

| BPred | Meaning |
|--------|--|
| 0b0000 | No branch predictor, or no MMU present. Implies a fixed MPU configuration. |
| 0b0001 | Branch predictor requires flushing on: <ul style="list-style-type: none"> Enabling or disabling a stage of address translation. Writing new data to instruction locations. Writing new mappings to the translation tables. Changes to the TTBR0, TTBR1, or TTBCR registers. Changes to the ContextID or ASID, or to the FCSE ProcessID if this is supported. |
| 0b0010 | Branch predictor requires flushing on: <ul style="list-style-type: none"> Enabling or disabling a stage of address translation. Writing new data to instruction locations. Writing new mappings to the translation tables. Any change to the TTBR0, TTBR1, or TTBCR registers without a change to the corresponding ContextID or ASID, or FCSE ProcessID if this is supported. |
| 0b0011 | Branch predictor requires flushing only on writing new data to instruction locations. |
| 0b0100 | For execution correctness, branch predictor requires no flushing at any time. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0010, 0b0011, or 0b0100. For values other than 0b0000 and 0b0100, the Arm Architecture Reference Manual, or the product documentation, might give more information about the required maintenance.

L1TstCln, bits [27:24]

Level 1 cache Test and Clean. Indicates the supported Level 1 data cache test and clean operations, for Harvard or unified cache implementations. Defined values are:

| L1TstCln | Meaning |
|-----------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported Level 1 data cache test and clean operations are: <ul style="list-style-type: none"> • Test and clean data cache. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Test, clean, and invalidate data cache. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1Uni, bits [23:20]

Level 1 Unified cache. Indicates the supported entire Level 1 cache maintenance operations for a unified cache implementation. Defined values are:

| L1Uni | Meaning |
|--------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported entire Level 1 cache operations are: <ul style="list-style-type: none"> • Invalidate cache, including branch predictor if appropriate. • Invalidate branch predictor, if appropriate. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Clean cache, using a recursive model that uses the cache dirty status bit. • Clean and invalidate cache, using a recursive model that uses the cache dirty status bit. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1Hvd, bits [19:16]

Level 1 Harvard cache. Indicates the supported entire Level 1 cache maintenance operations for a Harvard cache implementation. Defined values are:

| L1Hvd | Meaning |
|--------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported entire Level 1 cache operations are: <ul style="list-style-type: none"> • Invalidate instruction cache, including branch predictor if appropriate. • Invalidate branch predictor, if appropriate. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Invalidate data cache. • Invalidate data cache and instruction cache, including branch predictor if appropriate. |
| 0b0011 | As for 0b0010, and adds: <ul style="list-style-type: none"> • Clean data cache, using a recursive model that uses the cache dirty status bit. • Clean and invalidate data cache, using a recursive model that uses the cache dirty status bit. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1UniSW, bits [15:12]

Level 1 Unified cache by Set/Way. Indicates the supported Level 1 cache line maintenance operations by set/way, for a unified cache implementation. Defined values are:

| L1UniSW | Meaning |
|----------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported Level 1 unified cache line maintenance operations by set/way are: <ul style="list-style-type: none"> • Clean cache line by set/way. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Clean and invalidate cache line by set/way. |
| 0b0011 | As for 0b0010, and adds: <ul style="list-style-type: none"> • Invalidate cache line by set/way. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1HvdSW, bits [11:8]

Level 1 Harvard cache by Set/Way. Indicates the supported Level 1 cache line maintenance operations by set/way, for a Harvard cache implementation. Defined values are:

| L1HvdSW | Meaning |
|----------------|---|
| 0b0000 | None supported. |
| 0b0001 | Supported Level 1 Harvard cache line maintenance operations by set/way are: <ul style="list-style-type: none"> • Clean data cache line by set/way. • Clean and invalidate data cache line by set/way. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Invalidate data cache line by set/way. |
| 0b0011 | As for 0b0010, and adds: <ul style="list-style-type: none"> • Invalidate instruction cache line by set/way. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1UniVA, bits [7:4]

Level 1 Unified cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a unified cache implementation. Defined values are:

| L1UniVA | Meaning |
|----------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported Level 1 unified cache line maintenance operations by VA are: <ul style="list-style-type: none"> • Clean cache line by VA. • Invalidate cache line by VA. • Clean and invalidate cache line by VA. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Invalidate branch predictor by VA, if branch predictor is implemented. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1HvdVA, bits [3:0]

Level 1 Harvard cache by Virtual Address. Indicates the supported Level 1 cache line maintenance operations by VA, for a Harvard cache implementation. Defined values are:

| L1HvdVA | Meaning |
|---------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported Level 1 Harvard cache line maintenance operations by VA are: <ul style="list-style-type: none"> Clean data cache line by VA. Invalidate data cache line by VA. Clean and invalidate data cache line by VA. Clean instruction cache line by VA. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> Invalidate branch predictor by VA, if branch predictor is implemented. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Accessing ID_MMFR1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_MMFR1;
elsif PSTATE.EL == EL2 then
    return ID_MMFR1;
elsif PSTATE.EL == EL3 then
    return ID_MMFR1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR2, Memory Model Feature Register 2

The ID_MMFR2 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_MMFR2 bits [31:0] are architecturally mapped to AArch64 System register [ID_MMFR2_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_MMFR2 are UNDEFINED.

Attributes

ID_MMFR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------------|----|----|----|--------------------------|----|----|----|-------------------------|----|----|----|------------------------|----|----|----|------------------------|----|----|----|--------------------------|----|---|---|-------------------------|---|---|---|-------------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| HWAaccFlg | | | | WFIStall | | | | MemBarr | | | | UniTLB | | | | HvdTLB | | | | L1HvdRng | | | | L1HvdBG | | | | L1HvdFG | | | |

HWAaccFlg, bits [31:28]

Hardware Access Flag. In earlier versions of the Arm Architecture, this field indicates support for a Hardware Access flag, as part of the VMSAv7 implementation. Defined values are:

| HWAaccFlg | Meaning |
|-----------|--|
| 0b0000 | Not supported. |
| 0b0001 | Support for VMSAv7 Access flag, updated in hardware. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

WFIStall, bits [27:24]

Wait For Interrupt Stall. Indicates the support for Wait For Interrupt (WFI) stalling. Defined values are:

| WFIStall | Meaning |
|----------|---------------------------|
| 0b0000 | Not supported. |
| 0b0001 | Support for WFI stalling. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

MemBarr, bits [23:20]

Memory Barrier. Indicates the supported memory barrier System instructions in the (coproc == 1111) encoding space. Defined values are:

| MemBarr | Meaning |
|---------|---|
| 0b0000 | None supported. |
| 0b0001 | Supported memory barrier System instructions are: <ul style="list-style-type: none"> • Data Synchronization Barrier (DSB). |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Instruction Synchronization Barrier (ISB). • Data Memory Barrier (DMB). |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

Arm deprecates the use of these operations. [ID_ISAR4](#).Barrier_instrs indicates the level of support for the preferred barrier instructions.

UniTLB, bits [19:16]

Unified TLB. Indicates the supported TLB maintenance operations, for a unified TLB implementation. Defined values are:

| UniTLB | Meaning |
|--------|---|
| 0b0000 | Not supported. |
| 0b0001 | Supported unified TLB maintenance operations are: <ul style="list-style-type: none"> • Invalidate all entries in the TLB. • Invalidate TLB entry by VA. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Invalidate TLB entries by ASID match. |
| 0b0011 | As for 0b0010, and adds: <ul style="list-style-type: none"> • Invalidate instruction TLB and data TLB entries by VA All ASID. This is a shared unified TLB operation |
| 0b0100 | As for 0b0011, and adds: <ul style="list-style-type: none"> • Invalidate Hyp mode unified TLB entry by VA. • Invalidate entire Non-secure PL1&0 unified TLB. • Invalidate entire Hyp mode unified TLB. |
| 0b0101 | As for 0b0100, and adds the following operations: TLBIMVALIS , TLBIMVAALIS , TLBIMVALHIS , TLBIMVAL , TLBIMVAAL , TLBIMVALH . |
| 0b0110 | As for 0b0101, and adds the following operations: TLBIIPAS2IS , TLBIIPAS2LIS , TLBIIPAS2 , TLBIIPAS2L . |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0110.

HvdTLB, bits [15:12]

If the value of ID_MMFR2.UniTLB is not 0b0000, then the meaning of this field is IMPLEMENTATION DEFINED. Arm deprecates the use of this field by software.

L1HvdRng, bits [11:8]

Level 1 Harvard cache Range. Indicates the supported Level 1 cache maintenance range operations, for a Harvard cache implementation. Defined values are:

| L1HvdRng | Meaning |
|----------|--|
| 0b0000 | Not supported. |
| 0b0001 | Supported Level 1 Harvard cache maintenance range operations are: <ul style="list-style-type: none"> • Invalidate data cache range by VA. • Invalidate instruction cache range by VA. • Clean data cache range by VA. • Clean and invalidate data cache range by VA. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1HvdBG, bits [7:4]

Level 1 Harvard cache Background fetch. Indicates the supported Level 1 cache background fetch operations, for a Harvard cache implementation. When supported, background fetch operations are non-blocking operations. Defined values are:

| L1HvdBG | Meaning |
|----------------|--|
| 0b0000 | Not supported. |
| 0b0001 | Supported Level 1 Harvard cache background fetch operations are: <ul style="list-style-type: none"> Fetch instruction cache range by VA. Fetch data cache range by VA. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

L1HvdFG, bits [3:0]

Level 1 Harvard cache Foreground fetch. Indicates the supported Level 1 cache foreground fetch operations, for a Harvard cache implementation. When supported, foreground fetch operations are blocking operations. Defined values are:

| L1HvdFG | Meaning |
|----------------|--|
| 0b0000 | Not supported. |
| 0b0001 | Supported Level 1 Harvard cache foreground fetch operations are: <ul style="list-style-type: none"> Fetch instruction cache range by VA. Fetch data cache range by VA. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Accessing ID_MMFR2

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_MMFR2;
elsif PSTATE.EL == EL2 then
    return ID_MMFR2;
elsif PSTATE.EL == EL3 then
    return ID_MMFR2;

```


ID_MMFR3, Memory Model Feature Register 3

The ID_MMFR3 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_MMFR3 bits [31:0] are architecturally mapped to AArch64 System register [ID_MMFR3_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_MMFR3 are UNDEFINED.

Attributes

ID_MMFR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------------|----|----|----|------------------------|----|----|----|-------------------------|----|----|----|---------------------|----|----|----|---------------------------|----|----|----|-------------------------|----|---|---|--------------------------|---|---|---|--------------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Supersec | | | | CMemSz | | | | CohWalk | | | | PAN | | | | MaintBcst | | | | BPMaint | | | | CMaintSW | | | | CMaintVA | | | |

Supersec, bits [31:28]

Supersections. On a VMSA implementation, indicates whether Supersections are supported. Defined values are:

| Supersec | Meaning |
|----------|------------------------------|
| 0b0000 | Supersections supported. |
| 0b1111 | Supersections not supported. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b1111.

CMemSz, bits [27:24]

Cached Memory Size. Indicates the physical memory size supported by the caches. Defined values are:

| CMemSz | Meaning |
|--------|--|
| 0b0000 | 4GB, corresponding to a 32-bit physical address range. |
| 0b0001 | 64GB, corresponding to a 36-bit physical address range. |
| 0b0010 | 1TB or more, corresponding to a 40-bit or larger physical address range. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000, 0b0001, and 0b0010.

CohWalk, bits [23:20]

Coherent Walk. Indicates whether Translation table updates require a clean to the Point of Unification. Defined values are:

| CohWalk | Meaning |
|----------------|--|
| 0b0000 | Updates to the translation tables require a clean to the Point of Unification to ensure visibility by subsequent translation table walks. |
| 0b0001 | Updates to the translation tables do not require a clean to the Point of Unification to ensure visibility by subsequent translation table walks. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

PAN, bits [19:16]

Privileged Access Never. Indicates support for the PAN bit in [CPSR](#), [SPSR](#), and [DPSR](#) in AArch32 state. Defined values are:

| PAN | Meaning |
|------------|---|
| 0b0000 | PAN not supported. |
| 0b0001 | PAN supported. |
| 0b0010 | PAN supported and ATS1CPRP and ATS1CPWP instructions supported. |

All other values are reserved.

FEAT_PAN implements the functionality identified by the value 0b0001.

FEAT_PAN2 implements the functionality added by the value 0b0010.

In Armv8.1, the value 0b0000 is not permitted.

From Armv8.2, the only permitted value is 0b0010.

MaintBcst, bits [15:12]

Maintenance Broadcast. Indicates whether Cache, TLB, and branch predictor operations are broadcast. Defined values are:

| MaintBcst | Meaning |
|------------------|--|
| 0b0000 | Cache, TLB, and branch predictor operations only affect local structures. |
| 0b0001 | Cache and branch predictor operations affect structures according to shareability and defined behavior of instructions. TLB operations only affect local structures. |
| 0b0010 | Cache, TLB, and branch predictor operations affect structures according to shareability and defined behavior of instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

BPMaint, bits [11:8]

Branch Predictor Maintenance. Indicates the supported branch predictor maintenance operations in an implementation with hierarchical cache maintenance operations. Defined values are:

| BPMaint | Meaning |
|----------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported branch predictor maintenance operations are: <ul style="list-style-type: none"> • Invalidate all branch predictors. |
| 0b0010 | As for 0b0001, and adds: <ul style="list-style-type: none"> • Invalidate branch predictors by VA. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0010.

CMaintSW, bits [7:4]

Cache Maintenance by Set/Way. Indicates the supported cache maintenance operations by set/way, in an implementation with hierarchical caches. Defined values are:

| CMaintSW | Meaning |
|-----------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported hierarchical cache maintenance instructions by set/way are: <ul style="list-style-type: none"> • Invalidate data cache by set/way. • Clean data cache by set/way. • Clean and invalidate data cache by set/way. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

In a unified cache implementation, the data cache maintenance operations apply to the unified caches.

CMaintVA, bits [3:0]

Cache Maintenance by Virtual Address. Indicates the supported cache maintenance operations by VA, in an implementation with hierarchical caches. Defined values are:

| CMaintVA | Meaning |
|-----------------|--|
| 0b0000 | None supported. |
| 0b0001 | Supported hierarchical cache maintenance operations by VA are: <ul style="list-style-type: none"> • Invalidate data cache by VA. • Clean data cache by VA. • Clean and invalidate data cache by VA. • Invalidate instruction cache by VA. • Invalidate all instruction cache entries. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

In a unified cache implementation, data cache maintenance operations apply to the unified caches, and the instruction cache maintenance instructions are not implemented.

Accessing ID_MMFR3

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_MMFR3;
elsif PSTATE.EL == EL2 then
    return ID_MMFR3;
elsif PSTATE.EL == EL3 then
    return ID_MMFR3;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR4, Memory Model Feature Register 4

The ID_MMFR4 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_MMFR4 bits [31:0] are architecturally mapped to AArch64 System register [ID_MMFR4_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_MMFR4 are UNDEFINED.

Attributes

ID_MMFR4 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|----|----|-------|----|----|----|-----|----|----|----|------|----|----|----|-----|----|----|----|-----|----|---|---|-----|---|---|---|---------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EVT | | | | CCIDX | | | | LSM | | | | HPDS | | | | CnP | | | | XNX | | | | AC2 | | | | SpecSEI | | | |

EVT, bits [31:28]

Enhanced Virtualization Traps. If EL2 is implemented, indicates support for the [HCR2](#).{TTLBIS, TOCU, TICAB, TID4} traps. Defined values are:

| EVT | Meaning |
|--------|--|
| 0b0000 | HCR2 .{TTLBIS, TOCU, TICAB, TID4} traps are not supported. |
| 0b0001 | HCR2 .{TOCU, TICAB, TID4} traps are supported. |
| | HCR2 .TTLBIS trap is not supported. |
| 0b0010 | HCR2 .{TTLBIS, TOCU, TICAB, TID4} traps are supported. |

All other values are reserved.

FEAT_EVT implements the functionality identified by the values 0b0001 and 0b0010.

If EL2 is not implemented supporting AArch32, the only permitted value is 0b0000.

In Armv8.2, the permitted values are 0b0000, 0b0001, and 0b0010.

From Armv8.5, the permitted values are:

- 0b0000 when EL2 is not implemented or does not support AArch32.
- 0b0010 when EL2 is implemented and supports AArch32.

CCIDX, bits [27:24]

Support for use of the revised CCSIDR format and the presence of the CCSIDR2 is indicated. Defined values are:

| CCIDX | Meaning |
|--------|--|
| 0b0000 | 32-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is not implemented. |
| 0b0001 | 64-bit format implemented for all levels of the CCSIDR, and the CCSIDR2 register is implemented. |

All other values are reserved.

FEAT_CCIDX implements the functionality identified by 0b0001.

From Armv8.3, the permitted values are 0b0000 and 0b0001.

LSM, bits [23:20]

Indicates support for LSMAOE and nTLSMD bits in [HSCTLR](#) and [SCTLR](#). Defined values are:

| LSM | Meaning |
|--------|---------------------------------------|
| 0b0000 | LSMAOE and nTLSMD bits not supported. |
| 0b0001 | LSMAOE and nTLSMD bits supported. |

All other values are reserved.

FEAT_LSMAOC implements the functionality identified by the value 0b0001.

From Armv8.2, the permitted values are 0b0000 and 0b0001.

HPDS, bits [19:16]

Hierarchical permission disables bits in translation tables. Defined values are:

| HPDS | Meaning |
|--------|--|
| 0b0000 | Disabling of hierarchical controls not supported. |
| 0b0001 | Supports disabling of hierarchical controls using the TTBCR2 .HPD0, TTBCR2 .HPD1, and HTCR .HPD bits. |
| 0b0010 | As for value 0b0001, and adds possible hardware allocation of bits[62:59] of the translation table descriptors from the final lookup level for IMPLEMENTATION DEFINED use. |

All other values are reserved.

FEAT_AA32HPD implements the functionality identified by the value 0b0001.

FEAT_HPDS2 implements the functionality added by the value 0b0010.

Note

The value 0b0000 implies that the encoding for [TTBCR2](#) is UNDEFINED.

CnP, bits [15:12]

Common not Private translations. Defined values are:

| CnP | Meaning |
|--------|--|
| 0b0000 | Common not Private translations not supported. |
| 0b0001 | Common not Private translations supported. |

All other values are reserved.

FEAT_TTCNP implements the functionality identified by the value 0b0001.

From Armv8.2, the only permitted value is 0b0001.

XNX, bits [11:8]

Support for execute-never control distinction by Exception level at stage 2. Defined values are:

| XNX | Meaning |
|------------|---|
| 0b0000 | Distinction between EL0 and EL1 execute-never control at stage 2 not supported. |
| 0b0001 | Distinction between EL0 and EL1 execute-never control at stage 2 supported. |

All other values are reserved.

FEAT_XNX implements the functionality identified by the value 0b0001.

When FEAT_XNX is implemented:

- If all of the following conditions are true, it is IMPLEMENTATION DEFINED whether the value of ID_MMFR4.XNX is 0b0000 or 0b0001:
 - [ID_AA64MMFR1_EL1.XNX](#) == 1.
 - EL2 cannot use AArch32.
 - EL1 can use AArch32.
- If EL2 can use AArch32 then the only permitted value is 0b0001.

AC2, bits [7:4]

Indicates the extension of the [ACTLR](#) and [HACTLR](#) registers using [ACTLR2](#) and [HACTLR2](#). Defined values are:

| AC2 | Meaning |
|------------|---|
| 0b0000 | ACTLR2 and HACTLR2 are not implemented. |
| 0b0001 | ACTLR2 and HACTLR2 are implemented. |

All other values are reserved.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.2, the only permitted value is 0b0001.

SpecSEI, bits [3:0]

Describes whether the PE can generate SError interrupt exceptions from speculative reads of memory, including speculative instruction fetches. The defined values of this field are:

| SpecSEI | Meaning |
|----------------|--|
| 0b0000 | The PE never generates an SError interrupt due to an External abort on a speculative read. |
| 0b0001 | The PE might generate an SError interrupt due to an External abort on a speculative read. |

All other values are reserved.

Accessing ID_MMFR4

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_MMFR4) || boolean
IMPLEMENTATION_DEFINED "ID_MMFR4 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && (!IsZero(ID_MMFR4) || boolean
IMPLEMENTATION_DEFINED "ID_MMFR4 trapped by HCR.TID3") && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_MMFR4;
elsif PSTATE.EL == EL2 then
    return ID_MMFR4;
elsif PSTATE.EL == EL3 then
    return ID_MMFR4;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_MMFR5, Memory Model Feature Register 5

The ID_MMFR5 characteristics are:

Purpose

Provides information about the implemented memory model and memory management support in AArch32 state.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_MMFR5 bits [31:0] are architecturally mapped to AArch64 System register [ID_MMFR5_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_MMFR5 are UNDEFINED.

Attributes

ID_MMFR5 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|--------|---|---|-----|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | nTLBPA | | | ETS | | | | |

Bits [31:8]

Reserved, RES0.

nTLBPA, bits [7:4]

Indicates support for intermediate caching of translation table walks. Defined values are:

| nTLBPA | Meaning |
|--------|--|
| 0b0000 | The intermediate caching of translation table walks might include non-coherent caches of previous valid translation table entries since the last completed relevant TLBI applicable to the PE where either: <ul style="list-style-type: none"> The caching is indexed by the physical address of the location holding the translation table entry. The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry. |
| 0b0001 | The intermediate caching of translation table walks does not include non-coherent caches of previous valid translation table entries since the last completed TLBI applicable to the PE where either: <ul style="list-style-type: none"> The caching is indexed by the physical address of the location holding the translation table entry. The caching is used for stage 1 translations and is indexed by the intermediate physical address of the location holding the translation table entry. |

All other values are reserved.

FEAT_nTLBPA implements the functionality identified by the value 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

ETS, bits [3:0]

Indicates support for Enhanced Translation Synchronization. Defined values are:

| ETS | Meaning |
|--------|--|
| 0b0000 | Enhanced Translation Synchronization is not supported. |
| 0b0001 | Enhanced Translation Synchronization is supported. |

All other values are reserved.

FEAT_ETC implements the functionality identified by the value 0b0001.

From Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.7, the only permitted value is 0b0001.

Accessing ID_MMFR5

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0011 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && (!IsZero(ID_MMFR5) || boolean
IMPLEMENTATION_DEFINED "ID_MMFR5 trapped by HCR_EL2.TID3") && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && (!IsZero(ID_MMFR5) || boolean
IMPLEMENTATION_DEFINED "ID_MMFR5 trapped by HCR.TID3") && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_MMFR5;
elsif PSTATE.EL == EL2 then
    return ID_MMFR5;
elsif PSTATE.EL == EL3 then
    return ID_MMFR5;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_PFR0, Processor Feature Register 0

The ID_PFR0 characteristics are:

Purpose

Gives top-level information about the instruction sets and other features supported by the PE in AArch32 state.

Must be interpreted with [ID_PFR1](#).

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_PFR0 bits [31:0] are architecturally mapped to AArch64 System register [ID_PFR0_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_PFR0 are UNDEFINED.

Attributes

ID_PFR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|----|----|----|---------------------|----|----|----|---------------------|----|----|----|----------------------|----|----|----|------------------------|----|----|----|------------------------|----|---|---|------------------------|---|---|---|------------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RAS | | | | DIT | | | | AMU | | | | CSV2 | | | | State3 | | | | State2 | | | | State1 | | | | State0 | | | |

RAS, bits [31:28]

RAS Extension version. Defined values are:

| RAS | Meaning |
|--------|---|
| 0b0000 | No RAS Extension. |
| 0b0001 | RAS Extension implemented. |
| 0b0010 | FEAT_RASv1p1 implemented. As 0b0001, and adds support for additional ERXMISC<m> System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS and support for the optional RAS Timestamp Extension. |

All other values are reserved.

FEAT_RAS implements the functionality identified by the value 0b0001.

FEAT_RASv1p1 implements the functionality identified by the value 0b0010.

In Armv8.0 and Armv8.1, the permitted values are 0b0000 and 0b0001.

In Armv8.2, the only permitted value is 0b0001.

From Armv8.4, if FEAT_DoubleFault is implemented, the only permitted value is 0b0010.

From Armv8.4, when FEAT_DoubleFault is not implemented, and [ERRIDR.NUM](#) is 0, the permitted values are IMPLEMENTATION DEFINED 0b0001 or 0b0010.

Note

When the value of this field is 0b0001, [ID_PFR2.RAS_frac](#) indicates whether FEAT_RASv1p1 is implemented.

DIT, bits [27:24]

Data Independent Timing. Defined values are:

| DIT | Meaning |
|--------|---|
| 0b0000 | AArch32 does not guarantee constant execution time of any instructions. |
| 0b0001 | AArch32 provides the PSTATE.DIT mechanism to guarantee constant execution time of certain instructions. |

All other values are reserved.

FEAT_DIT implements the functionality identified by the value 0b0001.

From Armv8.4, the only permitted value is 0b0001.

AMU, bits [23:20]

Indicates support for Activity Monitors Extension. Defined values are:

| AMU | Meaning |
|--------|--|
| 0b0000 | Activity Monitors Extension is not implemented. |
| 0b0001 | FEAT_AMUv1 is implemented. |
| 0b0010 | FEAT_AMUv1p1 is implemented. As 0b0001 and adds support for virtualization of the activity monitor event counters. |

All other values are reserved.

FEAT_AMUv1 implements the functionality identified by the value 0b0001.

FEAT_AMUv1p1 implements the functionality identified by the value 0b0010.

In Armv8.0, the only permitted value is 0b0000.

In Armv8.4, the permitted values are 0b0000 and 0b0001.

From Armv8.6, the permitted values are 0b0000, 0b0001, and 0b0010.

CSV2, bits [19:16]

Speculative use of out of context branch targets. Defined values are:

| CSV2 | Meaning |
|--------|--|
| 0b0000 | This PE does not disclose whether branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context. |
| 0b0001 | Branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context only in a hard-to-determine way. |
| 0b0010 | Branch targets trained in one hardware-described context can exploitatively control speculative execution in a different hardware-described context only in a hard-to-determine way. Within a hardware-described context, branch targets trained for branches situated at one address can control speculative execution of branches situated at different addresses only in a hard-to-determine way. |

All other values are reserved.

FEAT_CSV2 implements the functionality identified by the values 0b0001 and 0b0010.

From Armv8.5, the permitted values are 0b0001 and 0b0010.

State3, bits [15:12]

T32EE instruction set support. Defined values are:

| State3 | Meaning |
|--------|------------------------------------|
| 0b0000 | Not implemented. |
| 0b0001 | T32EE instruction set implemented. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

State2, bits [11:8]

Jazelle extension support. Defined values are:

| State2 | Meaning |
|--------|---|
| 0b0000 | Not implemented. |
| 0b0001 | Jazelle extension implemented, without clearing of JOSCR.CV on exception entry. |
| 0b0010 | Jazelle extension implemented, with clearing of JOSCR.CV on exception entry. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

State1, bits [7:4]

T32 instruction set support. Defined values are:

| State1 | Meaning |
|--------|--|
| 0b0000 | T32 instruction set not implemented. |
| 0b0001 | T32 encodings before the introduction of Thumb-2 technology implemented: <ul style="list-style-type: none"> • All instructions are 16-bit. • A BL or BLX is a pair of 16-bit instructions. • 32-bit instructions other than BL and BLX cannot be encoded. |
| 0b0011 | T32 encodings after the introduction of Thumb-2 technology implemented, for all 16-bit and 32-bit T32 basic instructions. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0011.

State0, bits [3:0]

A32 instruction set support. Defined values are:

| State0 | Meaning |
|--------|--------------------------------------|
| 0b0000 | A32 instruction set not implemented. |
| 0b0001 | A32 instruction set implemented. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Accessing ID_PFR0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_PFR0;
elsif PSTATE.EL == EL2 then
    return ID_PFR0;
elsif PSTATE.EL == EL3 then
    return ID_PFR0;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_PFR1, Processor Feature Register 1

The ID_PFR1 characteristics are:

Purpose

Gives information about the AArch32 programmers' model.

Must be interpreted with [ID_PFR0](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_PFR1 bits [31:0] are architecturally mapped to AArch64 System register [ID_PFR1_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_PFR1 are UNDEFINED.

Attributes

ID_PFR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|----|----|----|---------------------------|----|----|----|--------------------------|----|----|----|--------------------------|----|----|----|--------------------------------|----|----|----|--------------------------|----|---|---|--------------------------|---|---|---|-------------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| GIC | | | | Virt_frac | | | | Sec_frac | | | | GenTimer | | | | Virtualization | | | | MProgMod | | | | Security | | | | ProgMod | | | |

GIC, bits [31:28]

System register GIC CPU interface. Defined values are:

| GIC | Meaning |
|--------|--|
| 0b0000 | GIC CPU interface system registers not implemented. |
| 0b0001 | System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported. |
| 0b0011 | System register interface to version 4.1 of the GIC CPU interface is supported. |

All other values are reserved.

Virt_frac, bits [27:24]

Virtualization fractional field. When the Virtualization field is 0b0000, determines the support for Virtualization Extensions. Defined values are:

| Virt_frac | Meaning |
|-----------|--|
| 0b0000 | No Virtualization Extensions are implemented. |
| 0b0001 | The following Virtualization Extensions are implemented: <ul style="list-style-type: none"> The SCR.SIF bit, if EL3 is implemented. The modifications to the SCR.AW and SCR.FW bits described in the Virtualization Extensions, if EL3 is implemented. The MSR (banked register) and MRS (banked register) instructions. The ERET instruction. |

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL2 is implemented.
- 0b0001 when EL2 is not implemented.

This field is valid only when the value of ID_PFR1.Virtualization is 0, otherwise it holds the value 0b0000.

Note

The ID_ISAR registers do not identify whether the instructions added by the Virtualization Extensions are implemented.

Sec_frac, bits [23:20]

Security fractional field. When the Security field is 0b0000, determines the support for Security Extensions. Defined values are:

| Sec_frac | Meaning |
|----------|--|
| 0b0000 | No Security Extensions are implemented. |
| 0b0001 | The following Security Extensions are implemented: <ul style="list-style-type: none"> • The VBAR register. • The TTBCR.PD0 and TTBCR.PD1 bits. |
| 0b0010 | As for 0b0001, plus the ability to access Secure or Non-secure physical memory is supported. |

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL3 is implemented.
- 0b0001 or 0b0010 when EL3 is not implemented.

This field is valid only when the value of ID_PFR1.Security is 0, otherwise it holds the value 0b0000.

GenTimer, bits [19:16]

Generic Timer support. Defined values are:

| GenTimer | Meaning |
|----------|--|
| 0b0000 | Generic Timer is not implemented. |
| 0b0001 | Generic Timer is implemented. |
| 0b0010 | Generic Timer is implemented, and also includes support for CNTHCTL .EVNTIS and CNTKCTL .EVNTIS fields, and CNTPCTSS and CNTVCTSS counter views. |

All other values are reserved.

FEAT_ECV implements the functionality identified by the value 0b0010.

In Armv8.0, the only permitted value is 0b0001.

From Armv8.6, the only permitted value is 0b0010.

Virtualization, bits [15:12]

Virtualization support. Defined values are:

| Virtualization | Meaning |
|----------------|--|
| 0b0000 | EL2, Hyp mode, and the HVC instruction not implemented. |
| 0b0001 | EL2, Hyp mode, the HVC instruction, and all the features described by Virt_frac == 0b0001 implemented. |

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL2 is not implemented.
- 0b0001 when EL2 is implemented.

In an implementation that includes EL2, if EL2 cannot use AArch32 but EL1 can use AArch32 then this field has the value 0b0001.

Note

The ID_ISARs do not identify whether the HVC instruction is implemented.

MProgMod, bits [11:8]

M-profile programmers' model support. Defined values are:

| MProgMod | Meaning |
|----------|---|
| 0b0000 | Not supported. |
| 0b0010 | Support for two-stack programmers' model. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Security, bits [7:4]

Security support. Defined values are:

| Security | Meaning |
|----------|--|
| 0b0000 | EL3, Monitor mode, and the SMC instruction not implemented. |
| 0b0001 | EL3, Monitor mode, the SMC instruction, and all the features described by Sec_frac == 0b0001 implemented. |
| 0b0010 | As for 0b0001, and adds the ability to set the NSACR .RFR bit. Not permitted in Armv8 as the NSACR .RFR bit is RES0. |

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 when EL3 is not implemented.
- 0b0001 when EL3 is implemented.

In an implementation that includes EL3, if EL3 cannot use AArch32 but EL1 can use AArch32 then this field has the value 0b0001.

ProgMod, bits [3:0]

Support for the standard programmers' model for ARMv4 and later. Model must support User, FIQ, IRQ, Supervisor, Abort, Undefined, and System modes. Defined values are:

| ProgMod | Meaning |
|---------|----------------|
| 0b0000 | Not supported. |
| 0b0001 | Supported. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0001.

Accessing ID_PFR1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_PFR1;
elsif PSTATE.EL == EL2 then
    return ID_PFR1;
elsif PSTATE.EL == EL3 then
    return ID_PFR1;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ID_PFR2, Processor Feature Register 2

The ID_PFR2 characteristics are:

Purpose

Gives information about the AArch32 programmers' model.

Must be interpreted with [ID_PFR0](#) and [ID_PFR1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register ID_PFR2 bits [31:0] are architecturally mapped to AArch64 System register [ID_PFR2_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ID_PFR2 are UNDEFINED.

Attributes

ID_PFR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|------|----|----|----|------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | RAS_frac | | | | SSBS | | | | CSV3 | | | | | | | | | | | |

Bits [31:12]

Reserved, RES0.

RAS_frac, bits [11:8]

RAS Extension fractional field.

| RAS_frac | Meaning |
|----------|---|
| 0b0000 | If ID_PFR0 .RAS == 0b0001, RAS Extension implemented. |
| 0b0001 | If ID_PFR0 .RAS == 0b0001, as 0b0000 and adds support for additional ERXMISC<m> System registers. Error records accessed through System registers conform to RAS System Architecture v1.1, which includes simplifications to ERR<n>STATUS and support for the optional RAS Timestamp Extension. |

All other values are reserved.

This field is valid only if [ID_PFR0](#).RAS == 0b0001.

SSBS, bits [7:4]

Speculative Store Bypassing controls in AArch64 state. Defined values are:

| SSBS | Meaning |
|--------|--|
| 0b0000 | AArch32 provides no mechanism to control the use of Speculative Store Bypassing. |
| 0b0001 | AArch32 provides the PSTATE.SSBS mechanism to mark regions that are Speculative Store Bypass Safe. |

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

All other values are reserved.

CSV3, bits [3:0]

Speculative use of faulting data. Defined values are:

| CSV3 | Meaning |
|--------|--|
| 0b0000 | This PE does not disclose whether data loaded under speculation with a permission or domain fault can be used to form an address or generate condition codes or SVE predicate values to be used by other instructions in the speculative sequence. |
| 0b0001 | Data loaded under speculation with a permission or domain fault cannot be used to form an address, generate condition codes, or generate SVE predicate values to be used by other instructions in the speculative sequence. The execution timing of any other instructions in the speculative sequence is not a function of the data loaded under speculation. |

All other values are reserved.

FEAT_CSV3 implements the functionality identified by the value 0b0001.

In Armv8.0, the permitted values are 0b0000 and 0b0001.

From Armv8.5, the only permitted value is 0b0001.

If FEAT_E0PD is implemented, FEAT_CSV3 must be implemented.

Accessing ID_PFR2

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0011 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ID_PFR2;
elsif PSTATE.EL == EL2 then
    return ID_PFR2;
elsif PSTATE.EL == EL3 then
    return ID_PFR2;

```


IFAR, Instruction Fault Address Register

The IFAR characteristics are:

Purpose

Holds the virtual address of the faulting address that caused a synchronous Prefetch Abort exception.

Configuration

AArch32 System register IFAR bits [31:0] are architecturally mapped to AArch64 System register [FAR_EL1\[63:32\]](#).

AArch32 System register IFAR bits [31:0] (S) are architecturally mapped to AArch32 System register [HIFAR\[31:0\]](#) when EL2 is implemented, EL3 is implemented and the implementation only supports execution in AArch32 state.

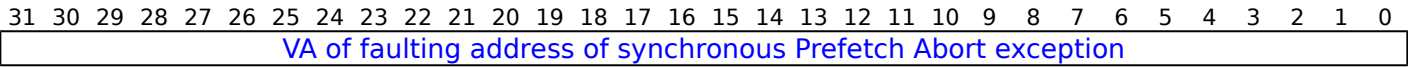
AArch32 System register IFAR bits [31:0] (S) are architecturally mapped to AArch64 System register [FAR_EL2\[63:32\]](#) when EL2 is implemented.

This register is present only when AArch32 is supported. Otherwise, direct accesses to IFAR are UNDEFINED.

Attributes

IFAR is a 32-bit register.

Field descriptions



Bits [31:0]

VA of faulting address of synchronous Prefetch Abort exception.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing IFAR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0110 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return IFAR_NS;
    else
        return IFAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return IFAR_NS;
    else
        return IFAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return IFAR_S;
    else
        return IFAR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0110 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T6 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T6 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        IFAR_NS = R[t];
    else
        IFAR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        IFAR_NS = R[t];
    else
        IFAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        IFAR_S = R[t];
    else
        IFAR_NS = R[t];

```

IFSR, Instruction Fault Status Register

The IFSR characteristics are:

Purpose

Holds status information about the last instruction fault.

Configuration

AArch32 System register IFSR bits [31:0] are architecturally mapped to AArch64 System register [IFSR32_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to IFSR are UNDEFINED.

The current translation table format determines which format of the register is used.

Attributes

IFSR is a 32-bit register.

Field descriptions

When TTBCR.EAE == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|------|-----|------|-------|------|------|---|---|---|---------|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | FnV | RES0 | ExT | RES0 | FS[4] | LPAE | RES0 | | | | FS[3:0] | | | | | | |

Bits [31:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|--|
| 0b0 | IFAR is valid. |
| 0b1 | IFAR is not valid, and holds an UNKNOWN value. |

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Prefetch Abort exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:13]

Reserved, RES0.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

FS, bits [10, 3:0]

Fault Status bits. Bits [10] and [3:0] are interpreted together.

| FS | Meaning | Applies when |
|---------|--|----------------------------------|
| 0b00001 | PC alignment fault. | |
| 0b00010 | Debug exception. | |
| 0b00011 | Access flag fault, level 1. | |
| 0b00101 | Translation fault, level 1. | |
| 0b00110 | Access flag fault, level 2. | |
| 0b00111 | Translation fault, level 2. | |
| 0b01000 | Synchronous External abort, not on translation table walk. | |
| 0b01001 | Domain fault, level 1. | |
| 0b01011 | Domain fault, level 2. | |
| 0b01100 | Synchronous External abort, on translation table walk, level 1. | |
| 0b01101 | Permission fault, level 1. | |
| 0b01110 | Synchronous External abort, on translation table walk, level 2. | |
| 0b01111 | Permission fault, level 2. | |
| 0b10000 | TLB conflict abort. | |
| 0b10100 | IMPLEMENTATION DEFINED fault (Lockdown fault). | |
| 0b11001 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b11100 | Synchronous parity or ECC error on translation table walk, level 1. | When FEAT_RAS is not implemented |
| 0b11110 | Synchronous parity or ECC error on translation table walk, level 2. | When FEAT_RAS is not implemented |

All other values are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Short-descriptor translation table lookup'.

The FS field is split as follows:

- FS[4] is IFSR[10].
- FS[3:0] is IFSR[3:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

| LPAE | Meaning |
|------|---|
| 0b0 | Using the Short-descriptor translation table formats. |
| 0b1 | Using the Long-descriptor translation table formats. |

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:4]

Reserved, RES0.

When TTBCR.EAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|------|-----|------|------|------|--------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | FnV | RES0 | ExT | RES0 | LPAE | RES0 | STATUS | | | | | | | | | |

Bits [31:17]

Reserved, RES0.

FnV, bit [16]

FAR not Valid, for a synchronous External abort other than a synchronous External abort on a translation table walk.

| FnV | Meaning |
|-----|--|
| 0b0 | IFAR is valid. |
| 0b1 | IFAR is not valid, and holds an UNKNOWN value. |

This field is valid only for a synchronous External abort other than a synchronous External abort on a translation table walk. It is RES0 for all other Prefetch Abort exceptions.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:13]

Reserved, RES0.

ExT, bit [12]

External abort type. This bit can be used to provide an IMPLEMENTATION DEFINED classification of External aborts.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

On taking a Data Abort exception, this bit is set as follows:

| LPAE | Meaning |
|------|---|
| 0b0 | Using the Short-descriptor translation table formats. |
| 0b1 | Using the Long-descriptor translation table formats. |

Hardware does not interpret this bit to determine the behavior of the memory system, and therefore software can set this bit to 0 or 1 without affecting operation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status bits. Possible values of this field are:

| STATUS | Meaning | Applies when |
|----------|--|----------------------------------|
| 0b000000 | Address size fault in translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010000 | Synchronous External abort, not on translation table walk. | |
| 0b010101 | Synchronous External abort on translation table walk, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk, level 3. | |
| 0b011000 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b011101 | Synchronous parity or ECC error on memory access on translation table walk, level 1. | When FEAT_RAS is not implemented |
| 0b011110 | Synchronous parity or ECC error on memory access on translation table walk, level 2. | When FEAT_RAS is not implemented |
| 0b011111 | Synchronous parity or ECC error on memory access on translation table walk, level 3. | When FEAT_RAS is not implemented |
| 0b100001 | PC alignment fault. | |
| 0b100010 | Debug exception. | |
| 0b110000 | TLB conflict abort. | |

All other values are reserved.

When FEAT_RAS is implemented, 0b011000, 0b011101, 0b011110, and 0b011111 are reserved.

For more information about the lookup level associated with a fault, see 'The level associated with MMU faults on a Long-descriptor translation table lookup'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing IFSR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return IFSR_NS;
    else
        return IFSR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return IFSR_NS;
    else
        return IFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return IFSR_S;
    else
        return IFSR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0101 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        IFSR_NS = R[t];
    else
        IFSR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        IFSR_NS = R[t];
    else
        IFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        IFSR_S = R[t];
    else
        IFSR_NS = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ISR, Interrupt Status Register

The ISR characteristics are:

Purpose

Shows the pending status of the IRQ, FIQ, or SError.

When executing at EL2, EL3, or Secure EL1, when [SCR_EL3.EEL2](#) == 0b0, this shows the pending status of the physical interrupts.

When executing at Non-secure EL1, or at Secure EL1, when [SCR_EL3.EEL2](#) == 0b01:

- If the [HCR](#).{IMO,FMO,AMO} bit has a value of 1, the corresponding ISR.{I,F,A} bit shows the pending status of the virtual IRQ, FIQ, or SError.
- If the [HCR](#).{IMO,FMO,AMO} bit has a value of 0, the corresponding ISR.{I,F,A} bit shows the pending status of the physical IRQ, FIQ, or SError.

Configuration

AArch32 System register ISR bits [31:0] are architecturally mapped to AArch64 System register [ISR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to ISR are UNDEFINED.

Attributes

ISR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | A | I | F | RES0 | | | | |

Bits [31:9]

Reserved, RES0.

A, bit [8]

SError interrupt pending bit:

| A | Meaning |
|-----|---------------------------------|
| 0b0 | No pending SError interrupt. |
| 0b1 | An SError interrupt is pending. |

If the SError interrupt is edge-triggered, this field is cleared to zero when the physical SError interrupt is taken.

I, bit [7]

IRQ pending bit. Indicates whether an IRQ interrupt is pending:

| I | Meaning |
|-----|------------------------------|
| 0b0 | No pending IRQ. |
| 0b1 | An IRQ interrupt is pending. |

F, bit [6]

FIQ pending bit. Indicates whether an FIQ interrupt is pending.

| F | Meaning |
|-----|------------------------------|
| 0b0 | No pending FIQ. |
| 0b1 | An FIQ interrupt is pending. |

Bits [5:0]

Reserved, RES0.

Accessing ISR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b0001 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return ISR;
elsif PSTATE.EL == EL2 then
    return ISR;
elsif PSTATE.EL == EL3 then
    return ISR;
```

ITLBIALL, Instruction TLB Invalidate All

The ITLBIALL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from instruction TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at EL1, all entries that:
 - Would be required for the EL1&0 translation regime.
 - Match the current VMID, if EL2 is implemented and enabled in the current Security state.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at EL2, and if EL2 is enabled in the current Security state, the stage 1 or stage 2 translation table entries that would be required for the Non-secure PL1&0 translation regime and matches the current VMID.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ITLBIALL are UNDEFINED.

Attributes

ITLBIALL is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the ITLBIALL instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0101 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
        && IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
            AArch32.ITLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_NSH,
            TLBI_ExcludeXS);
        else
            AArch32.ITLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_NSH, TLBI_AllAttr);
    endif PSTATE.EL == EL2 then
        AArch32.ITLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_NSH, TLBI_AllAttr);
    elsif PSTATE.EL == EL3 then
        AArch32.ITLBI_ALL(SecurityStateAtEL(EL3), Regime_EL30, Shareability_NSH, TLBI_AllAttr);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ITLBIASID, Instruction TLB Invalidate by ASID match

The ITLBIASID characteristics are:

Purpose

Invalidate all cached copies of translation table entries from instruction TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
 - Is from a level of lookup above the final level.
 - Is a non-global entry from the final level of lookup.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ITLBIASID are UNDEFINED.

Attributes

ITLBIASID is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | ASID | | | | | | | |

Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.

Executing the ITLBIASID instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| | | | | |
|--------|------|-----|-----|------|
| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0101 | 0b010 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
        && IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
            AArch32.ITLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBI_ExcludeXS, R[t]);
        else
            AArch32.ITLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBI_AllAttr, R[t]);
    endif PSTATE.EL == EL2 then
        AArch32.ITLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
        TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL3 then
        AArch32.ITLBI_ASID(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_NSH,
        TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ITLBIMVA, Instruction TLB Invalidate by VA

The ITLBIMVA characteristics are:

Purpose

Invalidate all cached copies of translation table entries from instruction TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
 - The entry is from a level of lookup above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Arm deprecates the use of this System instruction. It is only provided for backwards compatibility with earlier versions of the Arm architecture.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to ITLBIMVA are UNDEFINED.

Attributes

ITLBIMVA is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA | | | | | | | | | | | | RES0 | | | | ASID | | | | | | | | | | | | | | | |

VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this operation, regardless of the value of the ASID field.

Executing the ITLBIMVA instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0101 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
        && IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
            AArch32.ITLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBILevel_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.ITLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
            TLBILevel_Any, TLBI_AllAttr, R[t]);
        endif
    endif
elsif PSTATE.EL == EL2 then
    AArch32.ITLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH, TLBILevel_Any,
    TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.ITLBI_VA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_NSH,
    TLBILevel_Any, TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

JIDR, Jazelle ID Register

The JIDR characteristics are:

Purpose

A Jazelle register, which identified the Jazelle architecture version.

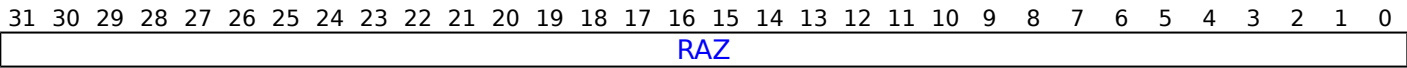
Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to JIDR are UNDEFINED.

Attributes

JIDR is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RAZ.

Accessing JIDR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b111 | 0b0000 | 0b0000 | 0b000 |

```
if PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JIDR UNDEFINED at EL0" then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HCR_EL2.TID0 == '1'
then
    AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID0 == '1' then
        AArch32.TakeHypTrapException(0x05);
    else
        return JIDR;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x05);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID0 == '1' then
        AArch32.TakeHypTrapException(0x05);
    else
        return JIDR;
elsif PSTATE.EL == EL2 then
    return JIDR;
elsif PSTATE.EL == EL3 then
    return JIDR;
```


JMCR, Jazelle Main Configuration Register

The JMCR characteristics are:

Purpose

A Jazelle register, which provides control of the Jazelle extension.

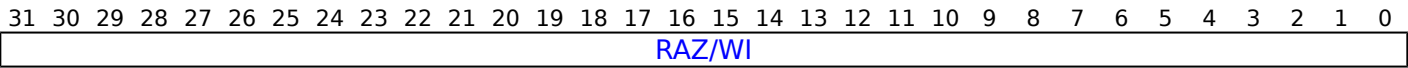
Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to JMCR are UNDEFINED.

Attributes

JMCR is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RAZ/WI.

Accessing JMCR

For accesses from EL0 it is IMPLEMENTATION DEFINED whether the register is RW or UNDEFINED.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b111 | 0b0010 | 0b0000 | 0b000 |

```
if PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JMCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        return JMCR;
elsif PSTATE.EL == EL1 then
    return JMCR;
elsif PSTATE.EL == EL2 then
    return JMCR;
elsif PSTATE.EL == EL3 then
    return JMCR;
```

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b111 | 0b0010 | 0b0000 | 0b000 |

```
if PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JMCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        //no operation
elseif PSTATE.EL == EL1 then
    //no operation
elseif PSTATE.EL == EL2 then
    //no operation
elseif PSTATE.EL == EL3 then
    //no operation
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

JOSCR, Jazelle OS Control Register

The JOSCR characteristics are:

Purpose

A Jazelle register, which provides operating system control of the Jazelle Extension.

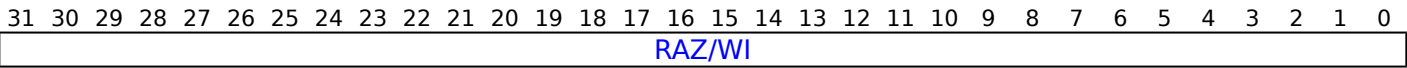
Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to JOSCR are UNDEFINED.

Attributes

JOSCR is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RAZ/WI.

Accessing JOSCR

Accesses to this register use the following encodings in the System register encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b111 | 0b0001 | 0b0000 | 0b000 |

```
if PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JOSCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        return JOSCR;
elsif PSTATE.EL == EL1 then
    return JOSCR;
elsif PSTATE.EL == EL2 then
    return JOSCR;
elsif PSTATE.EL == EL3 then
    return JOSCR;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1110 | 0b111 | 0b0001 | 0b0000 | 0b000 |

```
if PSTATE.EL == EL0 then
    if boolean IMPLEMENTATION_DEFINED "JOSCR UNDEFINED at EL0" then
        UNDEFINED;
    else
        //no operation
elseif PSTATE.EL == EL1 then
    //no operation
elseif PSTATE.EL == EL2 then
    //no operation
elseif PSTATE.EL == EL3 then
    //no operation
```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MAIR0, Memory Attribute Indirection Register 0

The MAIR0 characteristics are:

Purpose

Along with [MAIR1](#), provides the memory attribute encodings corresponding to the possible AttrIdx values in a Long-descriptor format translation table entry for stage 1 translations.

AttrIdx[2] indicates the MAIR register to be used:

- When AttrIdx[2] is 0, MAIR0 is used.
- When AttrIdx[2] is 1, [MAIR1](#) is used.

Configuration

AArch32 System register MAIR0 bits [31:0] are architecturally mapped to AArch64 System register [MAIR_EL1\[31:0\]](#) when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register MAIR0 bits [31:0] are architecturally mapped to AArch32 System register [PRRR\[31:0\]](#) when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register MAIR0 bits [31:0] (MAIR0_NS) are architecturally mapped to AArch32 System register [PRRR\[31:0\]](#) (PRRR_NS) when EL3 is using AArch32.

AArch32 System register MAIR0 bits [31:0] (MAIR0_S) are architecturally mapped to AArch32 System register [PRRR\[31:0\]](#) (PRRR_S) when EL3 is using AArch32.

This register is present only when AArch32 is supported. Otherwise, direct accesses to MAIR0 are UNDEFINED.

MAIR0 and [PRRR](#) are the same register, with a different view depending on the value of [TTBCR.EAE](#):

- When it is set to 0, the register is as described in [PRRR](#).
- When it is set to 1, the register is as described in MAIR0.

When EL3 is using AArch32, write access to MAIR0(S) is disabled when the CP15SDISABLE signal is asserted HIGH.

Attributes

MAIR0 is a 32-bit register.

Field descriptions

When TTBCR.EAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------|----|----|----|----|----|----|----|-----------------------|----|----|----|----|----|----|----|-----------------------|----|----|----|----|----|---|---|-----------------------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Attr3 | | | | | | | | Attr2 | | | | | | | | Attr1 | | | | | | | | Attr0 | | | | | | | |

Attr<n>, bits [8n+7:8n], for n = 3 to 0

The memory attribute encoding for an AttrIdx[2:0] entry in a Long descriptor format translation table entry, where:

- AttrIdx[2:0] gives the value of <n> in Attr<n>.
- AttrIdx[2] defines which MAIR to access. Attr7 to Attr4 are in MAIR1, and Attr3 to Attr0 are in MAIR0.

Bits [7:4] are encoded as follows:

| Attr<n>[7:4] | Meaning |
|------------------------|--|
| 0b0000 | Device memory. See encoding of Attr<n>[3:0] for the type of Device memory. |
| 0b00RW, RW not 0b00 | Normal memory, Outer Write-Through Transient. |
| 0b0100 | Normal memory, Outer Non-cacheable. |
| 0b01RW, RW not 0b00 | Normal memory, Outer Write-Back Transient. |
| 0b10RW | Normal memory, Outer Write-Through Non-transient. |
| 0b11RW | Normal memory, Outer Write-Back Non-transient. |

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

The meaning of bits [3:0] depends on the value of bits [7:4]:

| Attr<n>[3:0] | Meaning when Attr<n>[7:4] is 0b0000 | Meaning when Attr<n>[7:4] is not 0b0000 |
|------------------------|---|---|
| 0b0000 | Device-nGnRnE memory | UNPREDICTABLE |
| 0b00RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Through Transient |
| 0b0100 | Device-nGnRE memory | Normal memory, Inner Non-cacheable |
| 0b01RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Back Transient |
| 0b1000 | Device-nGRE memory | Normal memory, Inner Write-Through Non-transient (RW=0b00) |
| 0b10RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Through Non-transient |
| 0b1100 | Device-GRE memory | Normal memory, Inner Write-Back Non-transient (RW=0b00) |
| 0b11RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Back Non-transient |

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in some Attr<n> fields have the following meanings:

| R or W | Meaning |
|--------|-------------|
| 0b0 | No Allocate |
| 0b1 | Allocate |

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            return MAIR0_NS;
        else
            return PRRR_NS;
    else
        if TTBCR.EAE == '1' then
            return MAIR0;
        else
            return PRRR;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            if TTBCR.EAE == '1' then
                return MAIR0_NS;
            else
                return PRRR_NS;
        else
            if TTBCR.EAE == '1' then
                return MAIR0;
            else
                return PRRR;
    elsif PSTATE.EL == EL3 then
        if TTBCR.EAE == '1' then
            if SCR.NS == '0' then
                return MAIR0_S;
            else
                return MAIR0_NS;
        else
            if SCR.NS == '0' then
                return PRRR_S;
            else
                return PRRR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            MAIR0_NS = R[t];
        else
            PRRR_NS = R[t];
        else
            if TTBCR.EAE == '1' then
                MAIR0 = R[t];
            else
                PRRR = R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            if TTBCR.EAE == '1' then
                MAIR0_NS = R[t];
            else
                PRRR_NS = R[t];
        else
            if TTBCR.EAE == '1' then
                MAIR0 = R[t];
            else
                PRRR = R[t];
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' && CP15SDISABLE == HIGH then
            UNDEFINED;
        elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
            UNDEFINED;
        else
            if TTBCR.EAE == '1' then
                if SCR.NS == '0' then
                    MAIR0_S = R[t];
                else
                    MAIR0_NS = R[t];
            else
                if SCR.NS == '0' then
                    PRRR_S = R[t];
                else
                    PRRR_NS = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MAIR1, Memory Attribute Indirection Register 1

The MAIR1 characteristics are:

Purpose

Along with [MAIR0](#), provides the memory attribute encodings corresponding to the possible AttrIdx values in a Long-descriptor format translation table entry for stage 1 translations.

AttrIdx[2] indicates the MAIR register to be used:

- When AttrIdx[2] is 0, [MAIR0](#) is used.
- When AttrIdx[2] is 1, MAIR1 is used.

Configuration

AArch32 System register MAIR1 bits [31:0] are architecturally mapped to AArch64 System register [MAIR_EL1\[63:32\]](#) when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register MAIR1 bits [31:0] are architecturally mapped to AArch32 System register [NMRR\[31:0\]](#) when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register MAIR1 bits [31:0] (MAIR1_NS) are architecturally mapped to AArch32 System register [NMRR\[31:0\]](#) (NMRR_NS) when EL3 is using AArch32.

AArch32 System register MAIR1 bits [31:0] (MAIR1_S) are architecturally mapped to AArch32 System register [NMRR\[31:0\]](#) (NMRR_S) when EL3 is using AArch32.

This register is present only when AArch32 is supported. Otherwise, direct accesses to MAIR1 are UNDEFINED.

MAIR1 and [NMRR](#) are the same register, with a different view depending on the value of [TTBCR.EAE](#):

- When it is set to 0, the register is as described in [NMRR](#).
- When it is set to 1, the register is as described in MAIR1.

When EL3 is using AArch32, write access to MAIR1(S) is disabled when the CP15SDISABLE signal is asserted HIGH.

Attributes

MAIR1 is a 32-bit register.

Field descriptions

When TTBCR.EAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------|----|----|----|----|----|----|----|-----------------------|----|----|----|----|----|----|----|-----------------------|----|----|----|----|----|---|---|-----------------------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Attr7 | | | | | | | | Attr6 | | | | | | | | Attr5 | | | | | | | | Attr4 | | | | | | | |

Attr<n>, bits [8(n-4)+7:8(n-4)], for n = 7 to 4

The memory attribute encoding for an AttrIdx[2:0] entry in a Long descriptor format translation table entry, where:

- AttrIdx[2:0] gives the value of <n> in Attr<n>.
- AttrIdx[2] defines which MAIR to access. Attr7 to Attr4 are in MAIR1, and Attr3 to Attr0 are in MAIR0.

Bits [7:4] are encoded as follows:

| Attr<n>[7:4] | Meaning |
|------------------------|--|
| 0b0000 | Device memory. See encoding of Attr<n>[3:0] for the type of Device memory. |
| 0b00RW, RW not 0b00 | Normal memory, Outer Write-Through Transient. |
| 0b0100 | Normal memory, Outer Non-cacheable. |
| 0b01RW, RW not 0b00 | Normal memory, Outer Write-Back Transient. |
| 0b10RW | Normal memory, Outer Write-Through Non-transient. |
| 0b11RW | Normal memory, Outer Write-Back Non-transient. |

R = Outer Read-Allocate policy, W = Outer Write-Allocate policy.

The meaning of bits [3:0] depends on the value of bits [7:4]:

| Attr<n>[3:0] | Meaning when Attr<n>[7:4] is 0b0000 | Meaning when Attr<n>[7:4] is not 0b0000 |
|------------------------|---|---|
| 0b0000 | Device-nGnRnE memory | UNPREDICTABLE |
| 0b00RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Through Transient |
| 0b0100 | Device-nGnRE memory | Normal memory, Inner Non-cacheable |
| 0b01RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Back Transient |
| 0b1000 | Device-nGRE memory | Normal memory, Inner Write-Through Non-transient (RW=0b00) |
| 0b10RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Through Non-transient |
| 0b1100 | Device-GRE memory | Normal memory, Inner Write-Back Non-transient (RW=0b00) |
| 0b11RW, RW not 0b00 | UNPREDICTABLE | Normal memory, Inner Write-Back Non-transient |

R = Inner Read-Allocate policy, W = Inner Write-Allocate policy.

The R and W bits in some Attr<n> fields have the following meanings:

| R or W | Meaning |
|--------|-------------|
| 0b0 | No Allocate |
| 0b1 | Allocate |

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing MAIR1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1010 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            return MAIR1_NS;
        else
            return NMRR_NS;
    else
        if TTBCR.EAE == '1' then
            return MAIR1;
        else
            return NMRR;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            if TTBCR.EAE == '1' then
                return MAIR1_NS;
            else
                return NMRR_NS;
        else
            if TTBCR.EAE == '1' then
                return MAIR1;
            else
                return NMRR;
    elsif PSTATE.EL == EL3 then
        if TTBCR.EAE == '1' then
            if SCR.NS == '0' then
                return MAIR1_S;
            else
                return MAIR1_NS;
        else
            if SCR.NS == '0' then
                return NMRR_S;
            else
                return NMRR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1010 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            MAIR1_NS = R[t];
        else
            NMRR_NS = R[t];
    else
        if TTBCR.EAE == '1' then
            MAIR1 = R[t];
        else
            NMRR = R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            if TTBCR.EAE == '1' then
                MAIR1_NS = R[t];
            else
                NMRR_NS = R[t];
        else
            if TTBCR.EAE == '1' then
                MAIR1 = R[t];
            else
                NMRR = R[t];
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' && CP15SDISABLE == HIGH then
            UNDEFINED;
        elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
            UNDEFINED;
        else
            if TTBCR.EAE == '1' then
                if SCR.NS == '0' then
                    MAIR1_S = R[t];
                else
                    MAIR1_NS = R[t];
            else
                if SCR.NS == '0' then
                    NMRR_S = R[t];
                else
                    NMRR_NS = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MIDR, Main ID Register

The MIDR characteristics are:

Purpose

Provides identification information for the PE, including an implementer code for the device and a device ID number.

Configuration

AArch32 System register MIDR bits [31:0] are architecturally mapped to AArch64 System register [MIDR_EL1\[31:0\]](#).

AArch32 System register MIDR bits [31:0] are architecturally mapped to External register [MIDR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to MIDR are UNDEFINED.

Some fields of the MIDR are IMPLEMENTATION DEFINED. For more information about the values of these fields for a particular Armv8 implementation, and any implementation-specific significance of these values, see the product documentation.

Attributes

MIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|----|----|----|----|----|----|----|-------------------------|----|----|----|------------------------------|----|----|----|----|----|----|----|-------------------------|----|---|---|---|---|---|---|--------------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Implementer | | | | | | | | Variant | | | | Architecture | | | | | | | | PartNum | | | | | | | | Revision | | | |

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

| Implementer | Meaning |
|-------------|--|
| 0x00 | Reserved for software use. |
| 0x41 | Arm Limited. |
| 0x42 | Broadcom Corporation. |
| 0x43 | Cavium Inc. |
| 0x44 | Digital Equipment Corporation. |
| 0x46 | Fujitsu Ltd. |
| 0x49 | Infineon Technologies AG. |
| 0x4D | Motorola or Freescale Semiconductor Inc. |
| 0x4E | NVIDIA Corporation. |
| 0x50 | Applied Micro Circuits Corporation. |
| 0x51 | Qualcomm Inc. |
| 0x56 | Marvell International Ltd. |
| 0x69 | Intel Corporation. |
| 0xC0 | Ampere Computing. |

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

Access to this field is **RO**.

Variant, bits [23:20]

Variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Architecture, bits [19:16]

Architecture version. Defined values are:

| Architecture | Meaning |
|--------------|---|
| 0b0001 | Armv4. |
| 0b0010 | Armv4T. |
| 0b0011 | Armv5 (obsolete). |
| 0b0100 | Armv5T. |
| 0b0101 | Armv5TE. |
| 0b0110 | Armv5TEJ. |
| 0b0111 | Armv6. |
| 0b1111 | Architectural features are individually identified in the ID_* registers. |

All other values are reserved.

Access to this field is **RO**.

PartNum, bits [15:4]

Primary Part Number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [3:0]

Revision number for the device.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MIDR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b000 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) then
        return VPIDR_EL2<31:0>;
    elsif EL2Enabled() && ELUsingAArch32(EL2) then
        return VPIDR;
    else
        return MIDR;
elsif PSTATE.EL == EL2 then
    return MIDR;
elsif PSTATE.EL == EL3 then
    return MIDR;
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPIDR, Multiprocessor Affinity Register

The MPIDR characteristics are:

Purpose

In a multiprocessor system, provides an additional PE identification mechanism for scheduling purposes.

Configuration

AArch32 System register MPIDR bits [31:0] are architecturally mapped to AArch64 System register [MPIDR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to MPIDR are UNDEFINED.

In a uniprocessor system, Arm recommends that each Aff<n> field of this register returns a value of 0.

Attributes

MPIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|------|----|----|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| M | U | RES0 | | | | | MT | Aff2 | | | | | | | | Aff1 | | | | | | | | Aff0 | | | | | | | |

M, bit [31]

Indicates whether this implementation includes the functionality introduced by the Armv7 Multiprocessing Extensions.

| M | Meaning |
|-----|--|
| 0b0 | This implementation does not include the Armv7 Multiprocessing Extensions functionality. |
| 0b1 | This implementation includes the Armv7 Multiprocessing Extensions functionality. |

Access to this field is **RAO/WI**.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

| U | Meaning |
|-----|---|
| 0b0 | Processor is part of a multiprocessor system. |
| 0b1 | Processor is part of a uniprocessor system. |

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using a multithreading type approach. See the description of Aff0 for more information about affinity levels.

| MT | Meaning |
|-----|---|
| 0b0 | Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is largely independent. |
| 0b1 | Performance of PEs with different affinity level 0 values, and the same values for affinity level 1 and higher, is very interdependent. |

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

Aff0, bits [7:0]

Affinity level 0. This is the affinity level that is most significant for determining PE behavior. Higher affinity levels are increasingly less significant in determining PE behavior. The assigned value of the MPIDR.{Aff2, Aff1, Aff0} or [MPIDR_EL1](#).{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

Accessing MPIDR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) then
        return VMPIDR_EL2<31:0>;
    elseif EL2Enabled() && ELUsingAArch32(EL2) then
        return VMPIDR;
    else
        return MPIDR;
elseif PSTATE.EL == EL2 then
    return MPIDR;
elseif PSTATE.EL == EL3 then
    return MPIDR;

```

MVBAR, Monitor Vector Base Address Register

The MVBAR characteristics are:

Purpose

When EL3 is implemented and can use AArch32, holds the vector base address for any exception that is taken to Monitor mode.

Secure software must program the MVBAR with the required initial value as part of the PE boot sequence.

Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to MVBAR are UNDEFINED.

It is IMPLEMENTATION DEFINED whether MVBAR[0] has a fixed value and ignored writes, or takes the last value written to it.

On a Warm reset into EL3 using AArch32, the reset value of MVBAR is an IMPLEMENTATION DEFINED choice between the following:

- MVBAR[31:5] = an IMPLEMENTATION DEFINED value, which might be UNKNOWN, MVBAR[4:1] = RES0, and MVBAR[0] = 0.
- MVBAR[31:1] = an IMPLEMENTATION DEFINED value that is bits[31:1] of the AArch32 reset address, and MVBAR[0] = 1.

Attributes

MVBAR is a 32-bit register.

Field descriptions

When programmed with a vector base address:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|----------|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Vector Base Address | | | | | | | | | | | | | | | | | | | | | | | | | Reserved | | | | | | |

Bits [31:5]

Vector Base Address. Bits[31:5] of the base address of the exception vectors for exceptions taken to this Exception level. Bits[4:0] of an exception vector are the exception offset.

Reserved, bits [4:0]

Reserved, see Configurations.

Accessing MVBAR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsHighestEL(EL1) then
        return RVBAR;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if IsHighestEL(EL2) then
        return RVBAR;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    return MVBAR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        MVBAR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MVFR0, Media and VFP Feature Register 0

The MVFR0 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and Floating-point implementation.

Must be interpreted with [MVFR1](#) and [MVFR2](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register MVFR0 bits [31:0] are architecturally mapped to AArch64 System register [MVFR0_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to MVFR0 are UNDEFINED.

Implemented only if the implementation includes Advanced SIMD and floating-point instructions.

Attributes

MVFR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------|----|----|----|-------------------------|----|----|----|------------------------|----|----|----|--------------------------|----|----|----|------------------------|----|----|----|----------------------|----|---|---|----------------------|---|---|---|-------------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| FPRound | | | | FPShVec | | | | FPSqrt | | | | FPDivide | | | | FPTrap | | | | FPDP | | | | FPSP | | | | SIMDReg | | | |

FPRound, bits [31:28]

Floating-Point Rounding modes. Indicates whether the floating-point implementation provides support for rounding modes. Defined values are:

| FPRound | Meaning |
|---------|--|
| 0b0000 | Not implemented, or only Round to Nearest mode supported, except that Round towards Zero mode is supported for VCVT instructions that always use that rounding mode regardless of the FPSCR setting. |
| 0b0001 | All rounding modes supported. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

FPShVec, bits [27:24]

Short Vectors. Indicates whether the floating-point implementation provides support for the use of short vectors. Defined values are:

| FPShVec | Meaning |
|---------|-----------------------------------|
| 0b0000 | Short vectors not supported. |
| 0b0001 | Short vector operation supported. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

FPSqrt, bits [23:20]

Square Root. Indicates whether the floating-point implementation provides support for the ARMv6 VFP square root operations. Defined values are:

| FPSqrt | Meaning |
|---------------|----------------------------|
| 0b0000 | Not supported in hardware. |
| 0b0001 | Supported. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

The VSQRT.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VSQRT.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

FPDivide, bits [19:16]

Indicates whether the floating-point implementation provides support for VFP divide operations. Defined values are:

| FPDivide | Meaning |
|-----------------|----------------------------|
| 0b0000 | Not supported in hardware. |
| 0b0001 | Supported. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

The VDIV.F32 instruction also requires the single-precision floating-point attribute, bits [7:4], and the VDIV.F64 instruction also requires the double-precision floating-point attribute, bits [11:8].

FPTrap, bits [15:12]

Floating Point Exception Trapping. Indicates whether the floating-point implementation provides support for exception trapping. Defined values are:

| FPTrap | Meaning |
|---------------|----------------|
| 0b0000 | Not supported. |
| 0b0001 | Supported. |

All other values are reserved.

A value of 0b0001 indicates that, when the corresponding trap is enabled, a floating-point exception generates an exception.

FPDP, bits [11:8]

Double Precision. Indicates whether the floating-point implementation provides support for double-precision operations. Defined values are:

| FPDP | Meaning |
|-------------|--|
| 0b0000 | Not supported in hardware. |
| 0b0001 | Supported, VFPv2. |
| 0b0010 | Supported, VFPv3, VFPv4, or Armv8. VFPv3 and Armv8 add an instruction to load a double-precision floating-point constant, and conversions between double-precision and fixed-point values. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0010.

A value of 0b0001 or 0b0010 indicates support for all VFP double-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F64 is only available if the Square root field is 0b0001.

- VDIV.F64 is only available if the Divide field is 0b0001.
- Conversion between double-precision and single-precision is only available if the single-precision field is nonzero.

FPSP, bits [7:4]

Single Precision. Indicates whether the floating-point implementation provides support for single-precision operations. Defined values are:

| FPSP | Meaning |
|--------|---|
| 0b0000 | Not supported in hardware. |
| 0b0001 | Supported, VFPv2. |
| 0b0010 | Supported, VFPv3 or VFPv4. VFPv3 adds an instruction to load a single-precision floating-point constant, and conversions between single-precision and fixed-point values. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0010.

A value of 0b0001 or 0b0010 indicates support for all VFP single-precision instructions in the supported version of VFP, except that, in addition to this field being nonzero:

- VSQRT.F32 is only available if the Square root field is 0b0001.
- VDIV.F32 is only available if the Divide field is 0b0001.
- Conversion between double-precision and single-precision is only available if the double-precision field is nonzero.

SIMDReg, bits [3:0]

Advanced SIMD registers. Indicates whether the Advanced SIMD and floating-point implementation provides support for the Advanced SIMD and floating-point register bank. Defined values are:

| SIMDReg | Meaning |
|---------|--|
| 0b0000 | The implementation has no Advanced SIMD and floating-point support. |
| 0b0001 | The implementation includes floating-point support with 16 x 64-bit registers. |
| 0b0010 | The implementation includes Advanced SIMD and floating-point support with 32 x 64-bit registers. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0010.

Accessing MVFR0

Accesses to this register use the following encodings in the System register encoding space:

VMRS{<c>}{<q>} <Rt>, <spec_reg>

| reg |
|--------|
| 0b0111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x08);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
        else
            return MVFR0;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            UNDEFINED;
        elsif EL2Enabled() && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
HCPTR.TCP10 == '1') then
            AArch32.TakeHypTrapException(0x00);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x07);
            else
                return MVFR0;
    elsif PSTATE.EL == EL3 then
        if CPACR.cp10 == '00' then
            UNDEFINED;
        else
            return MVFR0;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MVFR1, Media and VFP Feature Register 1

The MVFR1 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and Floating-point implementation.

Must be interpreted with [MVFR0](#) and [MVFR2](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register MVFR1 bits [31:0] are architecturally mapped to AArch64 System register [MVFR1_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to MVFR1 are UNDEFINED.

Implemented only if the implementation includes Advanced SIMD and floating-point instructions.

Attributes

MVFR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------------|----|----|----|----------------------|----|----|----|------------------------|----|----|----|------------------------|----|----|----|-------------------------|----|----|----|------------------------|----|---|---|------------------------|---|---|---|-----------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SIMDFMAC | | | | FPHP | | | | SIMDHP | | | | SIMDSP | | | | SIMDInt | | | | SIMDLS | | | | FPDNaN | | | | FPFtZ | | | |

SIMDFMAC, bits [31:28]

Advanced SIMD Fused Multiply-Accumulate. Indicates whether the Advanced SIMD implementation provides fused multiply accumulate instructions. Defined values are:

| SIMDFMAC | Meaning |
|----------|------------------|
| 0b0000 | Not implemented. |
| 0b0001 | Implemented. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

The Advanced SIMD and floating-point implementations must provide the same level of support for these instructions.

FPHP, bits [27:24]

Floating Point Half Precision. Indicates the level of half-precision floating-point support. Defined values are:

| FPHP | Meaning |
|-------------|---|
| 0b0000 | Not supported. |
| 0b0001 | Floating-point half-precision conversion instructions are supported for conversion between single-precision and half-precision. |
| 0b0010 | As for 0b0001, and adds instructions for conversion between double-precision and half-precision. |
| 0b0011 | As for 0b0010, and adds support for half-precision floating-point arithmetic. |

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 in an implementation without floating-point support.
- 0b0010 in an implementation with floating-point support that does not include the FEAT_FP16 extension.
- 0b0011 in an implementation with floating-point support that includes the FEAT_FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the SIMDHP field, meaning the permitted values are:

| Half Precision instructions supported | FPHP | SIMDHP |
|--|-------------|---------------|
| No support | 0b0000 | 0b0000 |
| Conversions only | 0b0010 | 0b0001 |
| Conversions and arithmetic | 0b0011 | 0b0010 |

SIMDHP, bits [23:20]

Advanced SIMD Half Precision. Indicates the level of half-precision floating-point support. Defined values are:

| SIMDHP | Meaning |
|---------------|---|
| 0b0000 | Not supported. |
| 0b0001 | SIMD half-precision conversion instructions are supported for conversion between single-precision and half-precision. |
| 0b0010 | As for 0b0001, and adds support for half-precision floating-point arithmetic. |

All other values are reserved.

In Armv8-A, the permitted values are:

- 0b0000 in an implementation without SIMD floating-point support.
- 0b0001 in an implementation with SIMD floating-point support that does not include the FEAT_FP16 extension.
- 0b0010 in an implementation with SIMD floating-point support that includes the FEAT_FP16 extension.

The level of support indicated by this field must be equivalent to the level of support indicated by the FPHP field, meaning the permitted values are:

| Half Precision instructions supported | FPHP | SIMDHP |
|--|-------------|---------------|
| No support | 0b0000 | 0b0000 |
| Conversions only | 0b0010 | 0b0001 |
| Conversions and arithmetic | 0b0011 | 0b0010 |

SIMDSP, bits [19:16]

Advanced SIMD Single Precision. Indicates whether the Advanced SIMD and floating-point implementation provides single-precision floating-point instructions. Defined values are:

| SIMDSP | Meaning |
|---------------|---|
| 0b0000 | Not implemented. |
| 0b0001 | Implemented. This value is permitted only if the SIMDInt field is 0b0001. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

SIMDInt, bits [15:12]

Advanced SIMD Integer. Indicates whether the Advanced SIMD and floating-point implementation provides integer instructions. Defined values are:

| SIMDInt | Meaning |
|----------------|------------------|
| 0b0000 | Not implemented. |
| 0b0001 | Implemented. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

SIMDLS, bits [11:8]

Advanced SIMD Load/Store. Indicates whether the Advanced SIMD and floating-point implementation provides load/store instructions. Defined values are:

| SIMDLS | Meaning |
|---------------|------------------|
| 0b0000 | Not implemented. |
| 0b0001 | Implemented. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

FPDNaN, bits [7:4]

Default NaN mode. Indicates whether the floating-point implementation provides support only for the Default NaN mode. Defined values are:

| FPDNaN | Meaning |
|---------------|--|
| 0b0000 | Not implemented, or hardware supports only the Default NaN mode. |
| 0b0001 | Hardware supports propagation of NaN values. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

FPFtZ, bits [3:0]

Flush to Zero mode. Indicates whether the floating-point implementation provides support only for the Flush-to-Zero mode of operation. Defined values are:

| FPFtZ | Meaning |
|--------------|---|
| 0b0000 | Not implemented, or hardware supports only the Flush-to-Zero mode of operation. |
| 0b0001 | Hardware supports full denormalized number arithmetic. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0001.

Accessing MVFR1

Accesses to this register use the following encodings in the System register encoding space:

VMRS{<c>}{<q>} <Rt>, <spec_reg>

| reg |
|------------|
| 0b0110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x08);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
        else
            return MVFR1;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            UNDEFINED;
        elsif EL2Enabled() && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
HCPTR.TCP10 == '1') then
            AArch32.TakeHypTrapException(0x00);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x07);
            else
                return MVFR1;
    elsif PSTATE.EL == EL3 then
        if CPACR.cp10 == '00' then
            UNDEFINED;
        else
            return MVFR1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MVFR2, Media and VFP Feature Register 2

The MVFR2 characteristics are:

Purpose

Describes the features provided by the AArch32 Advanced SIMD and Floating-point implementation.

Must be interpreted with [MVFR0](#) and [MVFR1](#).

For general information about the interpretation of the ID registers see 'Principles of the ID scheme for fields in ID registers'.

Configuration

AArch32 System register MVFR2 bits [31:0] are architecturally mapped to AArch64 System register [MVFR2_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to MVFR2 are UNDEFINED.

Implemented only if the implementation includes Advanced SIMD and floating-point instructions.

Attributes

MVFR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|--------|---|---|---|----------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | FPMisc | | | | SIMDMisc | | | |

Bits [31:8]

Reserved, RES0.

FPMisc, bits [7:4]

Indicates whether the floating-point implementation provides support for miscellaneous VFP features.

| FPMisc | Meaning |
|--------|---|
| 0b0000 | Not implemented, or no support for miscellaneous features. |
| 0b0001 | Support for Floating-point selection. |
| 0b0010 | As 0b0001, and Floating-point Conversion to Integer with Directed Rounding modes. |
| 0b0011 | As 0b0010, and Floating-point Round to Integer Floating-point. |
| 0b0100 | As 0b0011, and Floating-point MaxNum and MinNum. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0100.

SIMDMisc, bits [3:0]

Indicates whether the Advanced SIMD implementation provides support for miscellaneous Advanced SIMD features.

| SIMDMisc | Meaning |
|-----------------|--|
| 0b0000 | Not implemented, or no support for miscellaneous features. |
| 0b0001 | Floating-point Conversion to Integer with Directed Rounding modes. |
| 0b0010 | As 0b0001, and Floating-point Round to Integer Floating-point. |
| 0b0011 | As 0b0010, and Floating-point MaxNum and MinNum. |

All other values are reserved.

In Armv8-A, the permitted values are 0b0000 and 0b0011.

Accessing MVFR2

Accesses to this register use the following encodings in the System register encoding space:

VMRS{<c>}{<q>} <Rt>, <spec_reg>

| reg |
|------------|
| 0b0101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        UNDEFINED;
    elsif (ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') || CPACR.cp10 == '00' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H != '1' && CPTR_EL2.TFP == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.E2H == '1' && CPTR_EL2.FPEN == 'x0' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x07);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && ((ELUsingAArch32(EL3) && SCR.NS == '1' &&
NSACR.cp10 == '0') || HCPTR.TCP10 == '1') then
        AArch32.TakeHypTrapException(0x08);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID3 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x08);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID3 == '1' then
        AArch32.TakeHypTrapException(0x08);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x07);
        else
            return MVFR2;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            UNDEFINED;
        elsif EL2Enabled() && ((ELUsingAArch32(EL3) && SCR.NS == '1' && NSACR.cp10 == '0') ||
HCPTR.TCP10 == '1') then
            AArch32.TakeHypTrapException(0x00);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && CPTR_EL3.TFP == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x07);
            else
                return MVFR2;
    elsif PSTATE.EL == EL3 then
        if CPACR.cp10 == '00' then
            UNDEFINED;
        else
            return MVFR2;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

NMRR, Normal Memory Remap Register

The NMRR characteristics are:

Purpose

Provides additional mapping controls for memory regions that are mapped as Normal memory by their entry in the [PRRR](#).

Used in conjunction with the [PRRR](#).

Configuration

AArch32 System register NMRR bits [31:0] are architecturally mapped to AArch64 System register [MAIR_EL1\[63:32\]](#) when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register NMRR bits [31:0] are architecturally mapped to AArch32 System register [MAIR1\[31:0\]](#) when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register NMRR bits [31:0] (NMRR_S) are architecturally mapped to AArch32 System register [MAIR1\[31:0\]](#) (MAIR1_S) when EL3 is using AArch32.

AArch32 System register NMRR bits [31:0] (NMRR_NS) are architecturally mapped to AArch32 System register [MAIR1\[31:0\]](#) (MAIR1_NS) when EL3 is using AArch32.

This register is present only when AArch32 is supported. Otherwise, direct accesses to NMRR are UNDEFINED.

[MAIR1](#) and NMRR are the same register, with a different view depending on the value of [TTBCR.EAE](#):

- When it is set to 0, the register is as described in NMRR.
- When it is set to 1, the register is as described in [MAIR1](#).

Attributes

NMRR is a 32-bit register.

Field descriptions

When TTBCR.EAE == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| OR7 | OR6 | OR5 | OR4 | OR3 | OR2 | OR1 | OR0 | IR7 | IR6 | IR5 | IR4 | IR3 | IR2 | IR1 | IR0 | | | | | | | | | | | | | | | | |

OR<n>, bits [2n+17:2n+16], for n = 7 to 0

Outer Cacheable property mapping for memory attributes n, if the region is mapped as Normal memory by the PRRR.TR<n> entry. n is the value of the TEX[0], C, and B bits concatenated.

| OR<n> | Meaning |
|-------|---|
| 0b00 | Region is Non-cacheable. |
| 0b01 | Region is Write-Back, Write-Allocate. |
| 0b10 | Region is Write-Through, no Write-Allocate. |
| 0b11 | Region is Write-Back, no Write-Allocate. |

The meaning of the field with n = 6 is IMPLEMENTATION DEFINED and might differ from the meaning given here. This is because the meaning of the attribute combination {TEX[0] = 1, C = 1, B = 0} is IMPLEMENTATION DEFINED.

When FEAT_XS is implemented, stage 1 Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IR<n>, bits [2n+1:2n], for n = 7 to 0

Inner Cacheable property mapping for memory attributes n, if the region is mapped as Normal memory by the PRRR.TR<n> entry. n is the value of the TEX[0], C, and B bits concatenated.

| IR<n> | Meaning |
|-------|---|
| 0b00 | Region is Non-cacheable. |
| 0b01 | Region is Write-Back, Write-Allocate. |
| 0b10 | Region is Write-Through, no Write-Allocate. |
| 0b11 | Region is Write-Back, no Write-Allocate. |

The meaning of the field with n = 6 is IMPLEMENTATION DEFINED and might differ from the meaning given here. This is because the meaning of the attribute combination {TEX[0] = 1, C = 1, B = 0} is IMPLEMENTATION DEFINED.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing NMRR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1010 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            return MAIR1_NS;
        else
            return NMRR_NS;
    else
        if TTBCR.EAE == '1' then
            return MAIR1;
        else
            return NMRR;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            if TTBCR.EAE == '1' then
                return MAIR1_NS;
            else
                return NMRR_NS;
        else
            if TTBCR.EAE == '1' then
                return MAIR1;
            else
                return NMRR;
    elsif PSTATE.EL == EL3 then
        if TTBCR.EAE == '1' then
            if SCR.NS == '0' then
                return MAIR1_S;
            else
                return MAIR1_NS;
        else
            if SCR.NS == '0' then
                return NMRR_S;
            else
                return NMRR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1010 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            MAIR1_NS = R[t];
        else
            NMRR_NS = R[t];
    else
        if TTBCR.EAE == '1' then
            MAIR1 = R[t];
        else
            NMRR = R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            if TTBCR.EAE == '1' then
                MAIR1_NS = R[t];
            else
                NMRR_NS = R[t];
        else
            if TTBCR.EAE == '1' then
                MAIR1 = R[t];
            else
                NMRR = R[t];
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' && CP15SDISABLE == HIGH then
            UNDEFINED;
        elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
            UNDEFINED;
        else
            if TTBCR.EAE == '1' then
                if SCR.NS == '0' then
                    MAIR1_S = R[t];
                else
                    MAIR1_NS = R[t];
            else
                if SCR.NS == '0' then
                    NMRR_S = R[t];
                else
                    NMRR_NS = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

NSACR, Non-Secure Access Control Register

The NSACR characteristics are:

Purpose

When EL3 is implemented and can use AArch32, defines the Non-secure access permissions to Trace, Advanced SIMD and floating-point functionality. Also includes IMPLEMENTATION DEFINED bits that can define Non-secure access permissions for IMPLEMENTATION DEFINED functionality.

Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to NSACR are UNDEFINED.

Note

In AArch64 state, the NSACR controls are replaced by controls in [CPTR_EL3](#).

Attributes

NSACR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----------|------|---------------------------|----|----------|------|------|------|------|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | NSTRCDIS | RES0 | IMPLEMENTATION DEFINED | | NSASEDIS | RES0 | cp11 | cp10 | RES0 | | | | | | | | | | | | |

If EL3 is implemented and is using AArch64 then:

- Any read of the NSACR from Non-secure EL2 or Non-secure EL1 returns a value of 0x00000C00.
- Any read or write to NSACR from Secure EL1 is trapped as an exception to EL3.

If EL3 is not implemented, then any read of the NSACR from EL2 or EL1 returns a value of 0x00000C00.

Bits [31:21]

Reserved, RES0.

NSTRCDIS, bit [20]

Disables Non-secure System register accesses to all implemented trace registers.

| NSTRCDIS | Meaning |
|----------|---|
| 0b0 | This control has no effect on: <ul style="list-style-type: none"> System register access to implemented trace registers. The behavior of CPACR.TRCDIS and HCPTR.TTA. |
| 0b1 | Non-secure System register accesses to all implemented trace registers are disabled, meaning: <ul style="list-style-type: none"> CPACR.TRCDIS behaves as RAO/WI in Non-secure state, regardless of its actual value. HCPTR.TTA behaves as RAO/WI, regardless of its actual value. |

The implementation of this field must correspond to the implementation of the [CPACR](#).TRCDIS field:

- If [CPACR](#).TRCDIS is RAZ/WI, this field is RAZ/WI.

- If [CPACR](#).TRCDIS is RW, this field is RW.

Note

- The ETMv4 architecture and ETE do not permit EL0 to access the trace registers. If the PE trace unit implements FEAT_ETMv4 or FEAT_ETE, EL0 accesses to the trace registers are UNDEFINED.
- The Arm architecture does not provide Non-secure access controls on trace register accesses through the optional memory-mapped external debug interface.

System register accesses to the trace registers can have side-effects. When a System register access is trapped, any side-effects that are normally associated with the access do not occur before the exception is taken.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Bit [19]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [18:16]

IMPLEMENTATION DEFINED.

NSASEDIS, bit [15]

Disables Non-secure access to the Advanced SIMD functionality.

| NSASEDIS | Meaning |
|----------|---|
| 0b0 | This control has no effect on: <ul style="list-style-type: none"> • Non-secure access to Advanced SIMD functionality. • The behavior of CPACR.ASEDIS and HCPTR.TASE. |
| 0b1 | Non-secure access to the Advanced SIMD functionality is disabled, meaning: <ul style="list-style-type: none"> • CPACR.ASEDIS behaves as RAO/WI in Non-secure state, regardless of its actual value. • HCPTR.TASE behaves as RAO/WI, regardless of its actual value. |

The implementation of this field must correspond to the implementation of the [CPACR](#).ASEDIS field:

- If [CPACR](#).ASEDIS is RES0, this field is RES0. If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.
- If [CPACR](#).ASEDIS is RAZ/WI, this field is RAZ/WI.
- If [CPACR](#).ASEDIS is RW, this field is RW.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Bits [14:12]

Reserved, RES0.

cp11, bit [11]

The value of this field is ignored. If this field is programmed with a different value to the cp10 field then this field is UNKNOWN on a direct read of the NSACR.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

cp10, bit [10]

Enable Non-secure access to the Advanced SIMD and floating-point features. Possible values of the fields are:

| cp10 | Meaning |
|------|---|
| 0b0 | Advanced SIMD and floating-point features can be accessed only from Secure state. Any attempt to access this functionality from Non-secure state is UNDEFINED. When the PE is in Non-secure state: <ul style="list-style-type: none">The CPACR.{cp11, cp10} fields ignore writes and read as 0b00, access denied.The HCPTR.{TCP11, TCP10} fields behave as RAO/WI, regardless of their actual values. |
| 0b1 | Advanced SIMD and floating-point features can be accessed from both Security states. |

If Non-secure access to the Advanced SIMD and floating-point functionality is enabled, the CPACR must be checked to determine the level of access that is permitted.

The Advanced SIMD and floating-point features controlled by these fields are:

- Execution of any floating-point or Advanced SIMD instruction.
- Any access to the Advanced SIMD and floating-point registers D0-D31 and their views as S0-S31 and Q0-Q15.
- Any access to the FPSCR, FPSID, MVFR0, MVFR1, MVFR2, or FPEXC System registers.

If the implementation does not include Advanced SIMD and floating-point functionality, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Bits [9:0]

Reserved, RES0.

Accessing NSACR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    elsif !HaveEL(EL3) || (!ELUsingAArch32(EL3) && SCR_EL3.NS == '1') then
        return Zeros(20):'1100':Zeros(8);
    else
        return NSACR;
elsif PSTATE.EL == EL2 then
    if !HaveEL(EL3) || (!ELUsingAArch32(EL3) && SCR_EL3.NS == '1') then
        return Zeros(20):'1100':Zeros(8);
    else
        return NSACR;
elsif PSTATE.EL == EL3 then
    return NSACR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        NSACR = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PAR, Physical Address Register

The PAR characteristics are:

Purpose

Returns the output address (OA) from an Address translation instruction that executed successfully, or fault information if the instruction did not execute successfully.

Configuration

AArch32 System register PAR bits [63:0] are architecturally mapped to AArch64 System register [PAR_EL1\[63:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to PAR are UNDEFINED.

PAR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits[31:0] and do not modify bits[63:32].

The Configurations section specifies the cases where each PAR format is used.

PAR is accessed as a 32-bit value:

- When the PE is not in Hyp mode and is using the Short-descriptor translation table format.
- When the PE is in Hyp mode and executes an [ATS12NSOPR](#), [ATS12NSOPW](#), [ATS12NSOUR](#), or [ATS12NSOUW](#) instruction and the value of [HCR.VM](#) is 0 and the value of [TTBCR.EAE](#) is 0.

In these cases, PAR[63:32] is RES0.

Otherwise, the PAR is accessed as a 64-bit value, if any of the following is true:

- When using the Long-descriptor translation table format.
- If the stage 1 address translation is disabled and [TTBCR.EAE](#) is set to 1.
- In an implementation that includes EL2, for the result of an ATS1Cxx instruction performed from Hyp mode.

For PL1&0 stage 1 translations, [TTBCR.EAE](#) selects the translation table format.

Attributes

PAR is a 64-bit register.

Field descriptions

When the instruction returned a 32-bit value to the PAR, PAR.F==0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|-----|----|----|----|---------------------------|----|----|----|----|------------|--|--|------------|--|----|---|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PA | | | | | | | | | | | | | | | | | | | | LPAEN | | NOS | | NS | | IMPLEMENTATION DEFINED | | | | SH | Inner[2:0] | | | Outer[1:0] | | SS | F |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | |

This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR can return a value that indicates the resulting attributes, rather than the values that appear in the translation table descriptors. More precisely:

- Memory attribute fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the translation table descriptors. This applies to the NOS, SH, Inner, and Outer fields.
- See the NS bit description for constraints on the value it returns.

Bits [63:32]

Reserved, RES0.

PA, bits [31:12]

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[31:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [11]

When updating the PAR with the result of the translation operation, this bit is set as follows:

| LPAE | Meaning |
|------|---|
| 0b0 | Short-descriptor translation table format used. This means the PAR returned a 32-bit value. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NOS, bit [10]

Not Outer Shareable. When the returned value of PAR.SH is 1, indicates the Shareability attribute for the physical memory region:

| NOS | Meaning |
|-----|-----------------------------------|
| 0b0 | Memory region is Outer Shareable. |
| 0b1 | Memory region is Inner Shareable. |

When the returned value of PAR.SH is 0 the value returned to this field is UNKNOWN.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [9]

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bit [8]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH, bit [7]

Shareability. Indicates whether the physical memory region is Non-shareable:

| SH | Meaning |
|-----------|--|
| 0b0 | Memory is Non-shareable. |
| 0b1 | Memory is shareable, and PAR.NOS indicates whether the region is Outer Shareable or Inner Shareable. |

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Inner[2:0], bits [6:4]

Inner cacheability attribute for the region. Permitted values are:

| Inner[2:0] | Meaning |
|-------------------|--------------------------------|
| 0b000 | Non-cacheable. |
| 0b001 | Device-nGnRnE. |
| 0b011 | Device-nGnRE. |
| 0b101 | Write-Back, Write-Allocate. |
| 0b110 | Write-Through. |
| 0b111 | Write-Back, no Write-Allocate. |

The values 0b010 and 0b100 are reserved.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Outer[1:0], bits [3:2]

Outer cacheability attribute for the region. Permitted values are:

| Outer[1:0] | Meaning |
|-------------------|-----------------------------------|
| 0b00 | Non-cacheable. |
| 0b01 | Write-Back, Write-Allocate. |
| 0b10 | Write-Through, no Write-Allocate. |
| 0b11 | Write-Back, no Write-Allocate. |

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SS, bit [1]

Supersection. Used to indicate if the result is a Supersection:

| SS | Meaning |
|-----------|--|
| 0b0 | Result is not a Supersection. PAR[31:12] contains OA[31:12]. |
| 0b1 | Result is a Supersection, and: <ul style="list-style-type: none"> PAR[31:24] contains OA[31:24]. PAR[23:16] contains OA[39:32]. PAR[15:12] contains 0b0000. If an implementation supports less than 40 bits of physical address, the bits in the PAR field that correspond to physical address bits that are not implemented are UNKNOWN. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

| F | Meaning |
|-----|---|
| 0b0 | Address translation completed successfully. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When the instruction returned a 32-bit value to the PAR, PAR.F==1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|------|----|----|----|-------|----|---------|----|----|----|---|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | RES0 | | | | LPAE | | RES0 | | | | FS[5] | | FS[4:0] | | | | F |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

This section describes the register value returned by a fault on the execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

Bits [63:32]

Reserved, RES0.

IMPLEMENTATION DEFINED, bits [31:16]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [15:12]

Reserved, RES0.

LPAE, bit [11]

When updating the PAR with the result of the translation operation, this bit is set as follows:

| LPAE | Meaning |
|------|---|
| 0b0 | Short-descriptor translation table format used. This means the PAR returned a 32-bit value. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [10:7]

Reserved, RES0.

FS[5], bit [6]

Fault status bits, External abort type. Provides an IMPLEMENTATION DEFINED classification of an External abort. Values are as in the [DESR.ExT](#) field when using the Short-descriptor translation table format.

In an implementation that does not provide any classification of External aborts, this bit is RES0.

For aborts other than External aborts this bit always returns 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FS[4:0], bits [5:1]

Fault status bits. Values are as in the [DEFSR.FS](#) field when using the Short-descriptor translation table format.

| FS[4:0] | Meaning | Applies when |
|---------|--|----------------------------------|
| 0b00001 | Alignment fault. | |
| 0b00011 | Access flag fault, level 1. | |
| 0b00100 | Fault on instruction cache maintenance. | |
| 0b00101 | Translation fault, level 1. | |
| 0b00110 | Access flag fault, level 2. | |
| 0b00111 | Translation fault, level 2. | |
| 0b01001 | Domain fault, level 1. | |
| 0b01011 | Domain fault, level 2. | |
| 0b01100 | Synchronous External abort, on translation table walk, level 1. | |
| 0b01101 | Permission fault, level 1. | |
| 0b01110 | Synchronous External abort, on translation table walk, level 2. | |
| 0b01111 | Permission fault, level 2. | |
| 0b10000 | TLB conflict abort. | |
| 0b11001 | Synchronous parity or ECC error on memory access, not on translation table walk. | When FEAT_RAS is not implemented |
| 0b11100 | Synchronous parity or ECC error on translation table walk, level 1. | When FEAT_RAS is not implemented |
| 0b11110 | Synchronous parity or ECC error on translation table walk, level 2. | When FEAT_RAS is not implemented |

The reset behavior of this field is:

- On a Warm reset, this field resets to an UNKNOWN value.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

| F | Meaning |
|-----|------------------------------|
| 0b1 | Address translation aborted. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When the instruction returned a 64-bit value to the PAR, PAR.F==0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|------|----|----|----|------|---------------------------|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ATTR | | | | | | | | RES0 | | | | | | | | | | | | | | | | PA | | | | | | | |
| PA | | | | | | | | | | | | LPAE | IMPLEMENTATION DEFINED | | | | NS | SH | RES0 | | | | | | F | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

This section describes the register value returned by the successful execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

On a successful conversion, the PAR can return a value that indicates the resulting attributes, rather than the values that appear in the translation table descriptors. More precisely:

- Memory attribute fields are permitted to report the resulting attributes, as determined by any permitted implementation choices and any applicable configuration bits, instead of reporting the values that appear in the translation table descriptors. This applies to the ATTR and SH fields.
- See the NS bit description for constraints on the value it returns.

ATTR, bits [63:56]

Memory attributes for the returned output address. This field uses the same encoding as the Attr<n> fields in [MAIR0](#) and [MAIR1](#).

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [55:40]

Reserved, RES0.

PA, bits [39:12]

Output address. The output address (OA) corresponding to the supplied input address. This field returns address bits[39:12].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [11]

When updating the PAR with the result of the translation operation, this bit is set as follows:

| LPAE | Meaning |
|------|--|
| 0b1 | Long-descriptor translation table format used. This means the PAR returned a 64-bit value. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bit [10]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS, bit [9]

Non-secure. The NS attribute for a translation table entry from a Secure translation regime.

For a result from a Secure translation regime, this bit reflects the Security state of the physical address space of the translation. This means it reflects the effect of the NSTable bits of earlier levels of the translation table walk if those NSTable bits have an effect on the translation.

For a result from a Non-secure translation regime, this bit is UNKNOWN.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SH, bits [8:7]

Shareability attribute, for the returned output address. Permitted values are:

| SH | Meaning |
|------|------------------|
| 0b00 | Non-shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

The value 0b01 is reserved.

Note

This field returns the value 0b10 for:

- Any type of Device memory.
- Normal memory with both Inner Non-cacheable and Outer Non-cacheable attributes.

The value returned in this field can be the resulting attribute, as determined by any permitted implementation choices and any applicable configuration bits, instead of the value that appears in the translation table descriptor.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [6:1]

Reserved, RES0.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

| F | Meaning |
|-----|---|
| 0b0 | Address translation completed successfully. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When the instruction returned a 64-bit value to the PAR, PAR.F==1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|------------------------|----|----|----|----|----|----|----|------------------------|----|------|----|--------|----|-------|----|------|----|-----|----|----|----|----|----|--|--|--|--|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | IMPLEMENTATION DEFINED | | | | | | | | IMPLEMENTATION DEFINED | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | LPAE | | RES0 | | FSTAGE | | S2WLK | | RES0 | | FST | | | F | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | |

This section describes the register value returned by a fault on the execution of an Address translation instruction. Software might subsequently write a different value to the register, and that write does not affect the operation of the PE.

IMPLEMENTATION DEFINED, bits [63:56]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [55:52]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [51:48]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [47:12]

Reserved, RES0.

LPAE, bit [11]

When updating the PAR with the result of the translation operation, this bit is set as follows:

| LPAE | Meaning |
|------|--|
| 0b1 | Long-descriptor translation table format used. This means the PAR returned a 64-bit value. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [10]

Reserved, RES0.

FSTAGE, bit [9]

Indicates the translation stage at which the translation aborted:

| FSTAGE | Meaning |
|--------|--|
| 0b0 | Translation aborted because of a fault in the stage 1 translation. |
| 0b1 | Translation aborted because of a fault in the stage 2 translation. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S2WLK, bit [8]

If this bit is set to 1, it indicates the translation aborted because of a stage 2 fault during a stage 1 translation table walk.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [7]

Reserved, RES0.

FST, bits [6:1]

Fault status field. Values are as in the [DFSR.STATUS](#) and [IFSR.STATUS](#) fields when using the Long-descriptor translation table format.

| FST | Meaning | Applies when |
|------------|--|----------------------------------|
| 0b000000 | Address size fault in translation table base register. | |
| 0b000001 | Address size fault, level 1. | |
| 0b000010 | Address size fault, level 2. | |
| 0b000011 | Address size fault, level 3. | |
| 0b000101 | Translation fault, level 1. | |
| 0b000110 | Translation fault, level 2. | |
| 0b000111 | Translation fault, level 3. | |
| 0b001001 | Access flag fault, level 1. | |
| 0b001010 | Access flag fault, level 2. | |
| 0b001011 | Access flag fault, level 3. | |
| 0b001101 | Permission fault, level 1. | |
| 0b001110 | Permission fault, level 2. | |
| 0b001111 | Permission fault, level 3. | |
| 0b010101 | Synchronous External abort on translation table walk, level 1. | |
| 0b010110 | Synchronous External abort on translation table walk, level 2. | |
| 0b010111 | Synchronous External abort on translation table walk, level 3. | |
| 0b011101 | Synchronous parity or ECC error on memory access on translation table walk, level 1. | When FEAT_RAS is not implemented |
| 0b011110 | Synchronous parity or ECC error on memory access on translation table walk, level 2. | When FEAT_RAS is not implemented |
| 0b011111 | Synchronous parity or ECC error on memory access on translation table walk, level 3. | When FEAT_RAS is not implemented |
| 0b110000 | TLB conflict abort. | |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [0]

Indicates whether the instruction performed a successful address translation.

| F | Meaning |
|----------|------------------------------|
| 0b1 | Address translation aborted. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PAR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b0111 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return PAR_NS<31:0>;
    else
        return PAR<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return PAR_NS<31:0>;
    else
        return PAR<31:0>;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return PAR_S<31:0>;
    else
        return PAR_NS<31:0>;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0111 | 0b0100 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        PAR_NS = ZeroExtend(R[t]);
    else
        PAR = ZeroExtend(R[t]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        PAR_NS = ZeroExtend(R[t]);
    else
        PAR = ZeroExtend(R[t]);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        PAR_S = ZeroExtend(R[t]);
    else
        PAR_NS = ZeroExtend(R[t]);

```

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b0111 | 0b0000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return PAR_NS;
    else
        return PAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return PAR_NS;
    else
        return PAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return PAR_S;
    else
        return PAR_NS;

```

MCCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b0111 | 0b0000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T7 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T7 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        PAR_NS = R[t2]:R[t];
    else
        PAR = R[t2]:R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        PAR_NS = R[t2]:R[t];
    else
        PAR = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        PAR_S = R[t2]:R[t];
    else
        PAR_NS = R[t2]:R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCCFILTR, Performance Monitors Cycle Count Filter Register

The PMCCFILTR characteristics are:

Purpose

Determines the modes in which the Cycle Counter, [PMCCNTR](#), increments.

Configuration

AArch32 System register PMCCFILTR bits [31:0] are architecturally mapped to AArch64 System register [PMCCFILTR_EL0\[31:0\]](#).

AArch32 System register PMCCFILTR bits [31:0] are architecturally mapped to External register [PMCCFILTR_EL0\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCCFILTR are UNDEFINED.

Attributes

PMCCFILTR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|-----|-----|-----|------|----|----|----|----|-----|------|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P | U | NSK | NSU | NSH | RES0 | | | | | RLU | RES0 | | | | | | | | | | | | | | | | | | | | |

P, bit [31]

Privileged filtering bit. Controls counting in EL1.
If EL3 is implemented, then counting in Non-secure EL1 is further controlled by the PMCCFILTR.NSK bit.

| P | Meaning |
|-----|-----------------------------|
| 0b0 | Count cycles in EL1. |
| 0b1 | Do not count cycles in EL1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

U, bit [30]

User filtering bit. Controls counting in EL0.
If EL3 is implemented, then counting in Non-secure EL0 is further controlled by the PMCCFILTR.NSU bit.
If FEAT_RME is implemented, then counting in Realm EL0 is further controlled by the PMCCFILTR.RLU bit.

| U | Meaning |
|-----|-----------------------------|
| 0b0 | Count cycles in EL0. |
| 0b1 | Do not count cycles in EL0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 (kernel) modes filtering bit. Controls counting in Non-secure EL1.

If the value of this bit is equal to the value of PMCCFILTR.P, cycles in Non-secure EL1 are counted.

Otherwise, cycles in Non-secure EL1 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When EL3 is implemented:

Non-secure EL0 (Unprivileged) filtering. Controls counting in Non-secure EL0.

If the value of this bit is equal to the value of PMCCFILTR.U, cycles in Non-secure EL0 are counted.

Otherwise, cycles in Non-secure EL0 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When EL2 is implemented:

EL2 (Hyp mode) filtering bit. Controls counting in EL2.

| NSH | Meaning |
|-----|-----------------------------|
| 0b0 | Do not count cycles in EL2. |
| 0b1 | Count cycles in EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bits [26:22]

Reserved, RES0.

RLU, bit [21]

When FEAT_RME is implemented:

Realm EL0 (unprivileged) filtering bit. Controls counting in Realm EL0.

If the value of this bit is equal to the value of the PMCCFILTR.U bit, cycles in Realm EL0 are counted.
Otherwise, cycles in Realm EL0 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [20:0]

Reserved, RES0.

Accessing PMCCFILTR

PMCCFILTR can also be accessed by using [PMXEVTYPER](#) with [PMSELR](#).SEL set to 0b11111.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b1111 | 0b111 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCCFILTR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCFILTR;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCFILTR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCFILTR;
elsif PSTATE.EL == EL3 then
    return PMCCFILTR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1110 | 0b1111 | 0b111 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elseif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.PMCCFILTR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCCFILTR = R[t];
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCCFILTR = R[t];
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCCFILTR = R[t];
elseif PSTATE.EL == EL3 then
    PMCCFILTR = R[t];

```

PMCCNTR, Performance Monitors Cycle Count Register

The PMCCNTR characteristics are:

Purpose

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles. See 'Time as measured by the Performance Monitors cycle counter' for more information.

[PMCCFILTER](#) determines the modes and states in which the PMCCNTR can increment.

Configuration

AArch32 System register PMCCNTR bits [63:0] are architecturally mapped to AArch64 System register [PMCCNTR_EL0\[63:0\]](#).

AArch32 System register PMCCNTR bits [63:0] are architecturally mapped to External register [PMCCNTR_EL0\[63:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCCNTR are UNDEFINED.

PMCCNTR is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits [31:0] and do not modify bits [63:32].

All counters are subject to any changes in clock frequency, including clock stopping caused by the WFI and WFE instructions. This means that it is CONSTRAINED UNPREDICTABLE whether or not PMCCNTR continues to increment when clocks are stopped by WFI and WFE instructions.

Attributes

PMCCNTR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | CCNT | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | CCNT | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

CCNT, bits [63:0]

Cycle count. Depending on the values of [PMCR](#).{LC,D}, this field increments in one of the following ways:

- Every processor clock cycle.
- Every 64th processor clock cycle.

Writing 1 to [PMCR](#).C sets this field to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCCNTR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1001 | 0b1101 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCNTR<31:0>;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCNTR<31:0>;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCCNTR<31:0>;
elsif PSTATE.EL == EL3 then
    return PMCCNTR<31:0>;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1001 | 0b1101 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCCNTR = ZeroExtend(R[t]);
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCCNTR = ZeroExtend(R[t]);
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCCNTR = ZeroExtend(R[t]);
elsif PSTATE.EL == EL3 then
    PMCCNTR = ZeroExtend(R[t]);

```

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|---------------|------------|-------------|
| 0b1111 | 0b1001 | 0b0000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif ELUsingAArch32(EL1) && PMUSERENR.<CR,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        return PMCCNTR;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        return PMCCNTR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        return PMCCNTR;
elsif PSTATE.EL == EL3 then
    return PMCCNTR;

```

MCCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|---------------|------------|-------------|
| 0b1111 | 0b1001 | 0b0000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x04);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x04);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCCNTR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        PMCCNTR = R[t2]:R[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        PMCCNTR = R[t2]:R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x04);
    else
        PMCCNTR = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    PMCCNTR = R[t2]:R[t];

```


PMCEID0, Performance Monitors Common Event Identification register 0

The PMCEID0 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range 0x0000 to 0x001F.

For more information about the Common events and the use of the PMCEIDn registers, see 'The PMU event number space and common events'.

Configuration

AArch32 System register PMCEID0 bits [31:0] are architecturally mapped to AArch64 System register [PMCEID0_EL0\[31:0\]](#).

AArch32 System register PMCEID0 bits [31:0] are architecturally mapped to External register [PMCEID0\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCEID0 are UNDEFINED.

Attributes

PMCEID0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|---------------------|---------------------|---------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 |
| ID31 | ID30 | ID29 | ID28 | ID27 | ID26 | ID25 | ID24 | ID23 | ID22 | ID21 | ID20 | ID19 | ID18 | ID17 | ID16 | ID15 | ID14 | ID13 | ID12 | ID11 | ID10 | ID9 | ID8 | ID7 |

ID<n>, bit [n], for n = 31 to 0

ID[n] corresponds to Common event n.

For each bit:

| ID<n> | Meaning |
|-------|--|
| 0b0 | The Common event is not implemented, or not counted. |
| 0b1 | The Common event is implemented. |

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU

features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b110 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCEID0;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCEID0;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCEID0;
elsif PSTATE.EL == EL3 then
    return PMCEID0;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID1, Performance Monitors Common Event Identification register 1

The PMCEID1 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range 0x0020 to 0x003F.

For more information about the Common events and the use of the PMCEIDn registers see 'The PMU event number space and common events'.

Configuration

AArch32 System register PMCEID1 bits [31:0] are architecturally mapped to AArch64 System register [PMCEID1_EL0\[31:0\]](#).

AArch32 System register PMCEID1 bits [31:0] are architecturally mapped to External register [PMCEID1\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCEID1 are UNDEFINED.

Attributes

PMCEID1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|---------------------|---------------------|---------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 |
| ID31 | ID30 | ID29 | ID28 | ID27 | ID26 | ID25 | ID24 | ID23 | ID22 | ID21 | ID20 | ID19 | ID18 | ID17 | ID16 | ID15 | ID14 | ID13 | ID12 | ID11 | ID10 | ID9 | ID8 | ID7 |

ID<n>, bit [n], for n = 31 to 0

ID[n] corresponds to Common event (0x0020 + n).

For each bit:

| ID<n> | Meaning |
|-------|--|
| 0b0 | The Common event is not implemented, or not counted. |
| 0b1 | The Common event is implemented. |

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU

features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b111 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCEID1;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCEID1;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCEID1;
elsif PSTATE.EL == EL3 then
    return PMCEID1;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID2, Performance Monitors Common Event Identification register 2

The PMCEID2 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range 0x4000 to 0x401F.

For more information about the Common events and the use of the PMCEIDn registers see 'The PMU event number space and common events'.

Configuration

AArch32 System register PMCEID2 bits [31:0] are architecturally mapped to AArch64 System register [PMCEID0_EL0\[63:32\]](#).

AArch32 System register PMCEID2 bits [31:0] are architecturally mapped to External register [PMCEID2\[63:32\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3p1 is implemented. Otherwise, direct accesses to PMCEID2 are UNDEFINED.

Attributes

PMCEID2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | |
|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 |
| IDhi31 | IDhi30 | IDhi29 | IDhi28 | IDhi27 | IDhi26 | IDhi25 | IDhi24 | IDhi23 | IDhi22 | IDhi21 | IDhi20 | IDhi19 | IDhi18 | IDhi17 | IDhi16 | IDhi15 |

IDhi<n>, bit [n], for n = 31 to 0

IDhi[n] corresponds to Common event (0x4000 + n).

For each bit:

| IDhi<n> | Meaning |
|---------|--|
| 0b0 | The Common event is not implemented, or not counted. |
| 0b1 | The Common event is implemented. |

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU

features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID2

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b100 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCEID2;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCEID2;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCEID2;
elsif PSTATE.EL == EL3 then
    return PMCEID2;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID3, Performance Monitors Common Event Identification register 3

The PMCEID3 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range 0x4020 to 0x403F.

For more information about the Common events and the use of the PMCEIDn registers see 'The PMU event number space and common events'.

Configuration

AArch32 System register PMCEID3 bits [31:0] are architecturally mapped to AArch64 System register [PMCEID1_EL0\[63:32\]](#).

AArch32 System register PMCEID3 bits [31:0] are architecturally mapped to External register [PMCEID3\[63:32\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3p1 is implemented. Otherwise, direct accesses to PMCEID3 are UNDEFINED.

Attributes

PMCEID3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | |
|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 |
| IDhi31 | IDhi30 | IDhi29 | IDhi28 | IDhi27 | IDhi26 | IDhi25 | IDhi24 | IDhi23 | IDhi22 | IDhi21 | IDhi20 | IDhi19 | IDhi18 | IDhi17 | IDhi16 | IDhi15 |

IDhi<n>, bit [n], for n = 31 to 0

IDhi[n] corresponds to Common event (0x4020 + n).

For each bit:

| IDhi<n> | Meaning |
|---------|--|
| 0b0 | The Common event is not implemented, or not counted. |
| 0b1 | The Common event is implemented. |

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU

features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID3

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b101 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCEIDn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCEID3;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCEID3;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCEID3;
elsif PSTATE.EL == EL3 then
    return PMCEID3;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCNTENCLR, Performance Monitors Count Enable Clear register

The PMCNTENCLR characteristics are:

Purpose

Disables the Cycle Count Register, [PMCCNTR](#), and any implemented event counters [PMEVCNTR<n>](#). Reading this register shows which counters are enabled.

PMCNTENCLR is used in conjunction with the [PMCNTENSET](#) register.

Configuration

AArch32 System register PMCNTENCLR bits [31:0] are architecturally mapped to AArch64 System register [PMCNTENCLR_ELO\[31:0\]](#).

AArch32 System register PMCNTENCLR bits [31:0] are architecturally mapped to External register [PMCNTENCLR_ELO\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCNTENCLR are UNDEFINED.

Attributes

PMCNTENCLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

C, bit [31]

[PMCCNTR](#) disable bit. Disables the cycle counter register.

| C | Meaning |
|-----|--|
| 0b0 | When read, means the cycle counter is disabled. When written, has no effect. |
| 0b1 | When read, means the cycle counter is enabled. When written, disables the cycle counter. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter disable bit for [PMEVCNTR<n>](#).

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in [MDCR_EL2](#).HPMN if EL2 is using AArch64, or in [HDCR](#).HPMN if EL2 is using AArch32. Otherwise, N is the value in [PMCR](#).N.

| P<n> | Meaning |
|-------------------|--|
| 0b0 | When read, means that PMEVCNTR<n> is disabled. When written, has no effect. |
| 0b1 | When read, means that PMEVCNTR<n> is enabled. When written, disables PMEVCNTR<n> . |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCNTENCLR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b010 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCNTEN == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCNTENCLR;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCNTENCLR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCNTENCLR;
elsif PSTATE.EL == EL3 then
    return PMCNTENCLR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b010 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENCLR = R[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENCLR = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENCLR = R[t];
elsif PSTATE.EL == EL3 then
    PMCNTENCLR = R[t];

```


PMCNTENSET, Performance Monitors Count Enable Set register

The PMCNTENSET characteristics are:

Purpose

Enables the Cycle Count Register, [PMCCNTR](#), and any implemented event counters [PMEVCNTR<n>](#). Reading this register shows which counters are enabled.

PMCNTENSET is used in conjunction with the [PMCNTENCLR](#) register.

Configuration

AArch32 System register PMCNTENSET bits [31:0] are architecturally mapped to AArch64 System register [PMCNTENSET_EL0\[31:0\]](#).

AArch32 System register PMCNTENSET bits [31:0] are architecturally mapped to External register [PMCNTENSET_EL0\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCNTENSET are UNDEFINED.

Attributes

PMCNTENSET is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

C, bit [31]

[PMCCNTR](#) enable bit. Enables the cycle counter register.

| C | Meaning |
|-----|---|
| 0b0 | When read, means the cycle counter is disabled. When written, has no effect. |
| 0b1 | When read, means the cycle counter is enabled. When written, enables the cycle counter. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter enable bit for [PMEVCNTR<n>](#).

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in [MDCR_EL2](#).HPMN if EL2 is using AArch64, or in [HDCR](#).HPMN if EL2 is using AArch32. Otherwise, N is the value in [PMCR](#).N.

| P<n> | Meaning |
|-------------------|---|
| 0b0 | When read, means that PMEVCNTR<n> is disabled. When written, has no effect. |
| 0b1 | When read, means that PMEVCNTR<n> event counter is enabled. When written, enables PMEVCNTR<n> . |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCNTENSET

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMCNTEN == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCNTENSET;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCNTENSET;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCNTENSET;
elsif PSTATE.EL == EL3 then
    return PMCNTENSET;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCNTEN == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENSET = R[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENSET = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCNTENSET = R[t];
elsif PSTATE.EL == EL3 then
    PMCNTENSET = R[t];

```


PMCR, Performance Monitors Control Register

The PMCR characteristics are:

Purpose

Provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

Configuration

AArch32 System register PMCR bits [31:0] are architecturally mapped to AArch64 System register [PMCR_EL0\[31:0\]](#).

AArch32 System register PMCR bits [7:0] are architecturally mapped to External register [PMCR_EL0\[7:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMCR are UNDEFINED.

Attributes

PMCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|------|----|------|----|----|----|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMP | | | | | | | | IDCODE | | | | | | | | N | | | | RES0 | FZ | RES0 | LP | LC | DP | X | D | C | P | E | |

IMP, bits [31:24]

When FEAT_PMUv3p7 is not implemented:

Implementer code.

If this field is zero, then PMCR.IDCODE is RES0 and software must use [MIDR](#) to identify the PE.

Otherwise, this field and PMCR.IDCODE identify the PMU implementation to software. The implementer codes are allocated by Arm. A non-zero value has the same interpretation as [MIDR](#).Implementer.

Use of this field is deprecated.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RAZ.

IDCODE, bits [23:16]

When PMCR.IMP != 0b00000000:

Identification code. Use of this field is deprecated.

Each implementer must maintain a list of identification codes that are specific to the implementer. A specific implementation is identified by the combination of the implementer code and the identification code.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

N, bits [15:11]

Indicates the number of event counters implemented. This value is in the range of 0b000000-0b11111. If the value is 0b000000, then only [PMCCNTR](#) is implemented. If the value is 0b11111, then [PMCCNTR](#) and 31 event counters are implemented.

In an implementation that includes EL2:

- If EL2 is using AArch32, reads of this field from Non-secure EL1 and Non-secure EL0 return the value of [HDCR](#).HPMN.
- If EL2 is using AArch64 and is enabled in the current Security state, reads of this field from EL1 and EL0 return the value of [MDCR_EL2](#).HPMN.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [10]

Reserved, RES0.

FZO, bit [9]

When FEAT_PMUv3p7 is implemented:

Freeze-on-overflow. Stop event counters on overflow.

In the description of this field:

- If EL2 is implemented and is using AArch32, PMN is [HDCR](#).HPMN.
- If EL2 is implemented and is using AArch64, PMN is [MDCR_EL2](#).HPMN.
- If EL2 is not implemented, PMN is PMCR.N.

| FZO | Meaning |
|-----|--|
| 0b0 | Do not freeze on overflow. |
| 0b1 | Event counter PMEVCNTR<n> does not count when PMOVSr[(PMN-1):0] is nonzero and n is in the range of affected event counters. |

If PMN is not 0, this field affects the operation of event counters in the range [0 .. (PMN-1)].

This field does not affect the operation of other event counters and [PMCCNTR](#)

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [8]

Reserved, RES0.

LP, bit [7]**When FEAT_PMUv3p5 is implemented:**

Long event counter enable. Determines when unsigned overflow is recorded by an event counter overflow bit.

In the description of this field:

- If EL2 is implemented and is using AArch32, PMN is [HDCR](#).HPMN.
- If EL2 is implemented and is using AArch64, PMN is [MDCR_EL2](#).HPMN.
- If EL2 is not implemented, PMN is PMCR.N.

| LP | Meaning |
|-----|--|
| 0b0 | Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n> [31:0]. |
| 0b1 | Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n> [63:0]. |

If the highest implemented Exception level is using AArch32, it is IMPLEMENTATION DEFINED whether this bit is RW or RAZ/WI.

If PMN is not 0, this bit affects the operation of event counters in the range [0 .. (PMN-1)].

This field does not affect the operation of other event counters and [PMCCNTR](#).

[PMEVCNTR<n>](#)[63:32] cannot be accessed directly in AArch32 state.

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

LC, bit [6]

Long cycle counter enable. Determines when unsigned overflow is recorded by the cycle counter overflow bit.

| LC | Meaning |
|-----|--|
| 0b0 | Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR [31:0]. |
| 0b1 | Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR [63:0]. |

Arm deprecates use of [PMCR](#).LC = 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DP, bit [5]**When EL3 is implemented or (FEAT_PMUv3p1 is implemented and EL2 is implemented):**

Disable cycle counter when event counting is prohibited.

| DP | Meaning |
|-----|---|
| 0b0 | Cycle counting by PMCCNTR is not affected by this mechanism. |
| 0b1 | Cycle counting by PMCCNTR is disabled in prohibited regions: <ul style="list-style-type: none"> If FEAT_PMUv3p1 is implemented, EL2 is implemented, and HDCR.HPMD is 1, then cycle counting by PMCCNTR is disabled at EL2. If FEAT_PMUv3p7 is implemented, EL3 is implemented and using AArch64, and MDCR_EL3.MPMX is 1, then cycle counting by PMCCNTR is disabled at EL3. If EL3 is implemented, MDCR_EL3.SPME or SDCR.SPME is 0, and either FEAT_PMUv3p7 is not implemented, EL3 is using AArch32, or MDCR_EL3.MPMX is 0, then cycle counting by PMCCNTR is disabled at EL3 and in Secure state. <p>If HDCR.HPMN is not 0, this is when event counting by event counters in the range [0..HDCR.HPMN-1] is prohibited.</p> |

For more information see 'Prohibiting event counting'

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RES0.

X, bit [4]

When the implementation includes a PMU event export bus:

Enable export of events in an IMPLEMENTATION DEFINED PMU event export bus.

| X | Meaning |
|-----|-------------------------------------|
| 0b0 | Do not export events. |
| 0b1 | Export events where not prohibited. |

This field enables the exporting of events over an IMPLEMENTATION DEFINED PMU event export bus to another device, for example to an OPTIONAL PE trace unit.

No events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Otherwise:

Reserved, RAZ/WI.

D, bit [3]

Clock divider.

| D | Meaning |
|-----|--|
| 0b0 | When enabled, PMCCNTR counts every clock cycle. |
| 0b1 | When enabled, PMCCNTR counts once every 64 clock cycles. |

If PMCR.LC == 1, this bit is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR.D = 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

C, bit [2]

Cycle counter reset. The effects of writing to this bit are:

| C | Meaning |
|-----|--|
| 0b0 | No action. |
| 0b1 | Reset PMCCNTR to zero. |

Note

Resetting [PMCCNTR](#) does not change the cycle counter overflow bit. If FEAT_PMUv3p5 is implemented, the value of PMCR.LC is ignored, and bits [63:0] of the cycle counter are reset.

Access to this field is **WO/RAZ**.

P, bit [1]

Event counter reset.

In the description of this field:

- If EL2 is implemented and is using AArch32, PMN is [HDCR](#).HPMN.
- If EL2 is implemented and is using AArch64, PMN is [MDCR_EL2](#).HPMN.
- If EL2 is not implemented, PMN is PMCR.N.

| P | Meaning |
|-----|---|
| 0b0 | No action. |
| 0b1 | If n is in the range of affected event counters, resets each event counter PMEVCNTR<n> to zero. |

The effects of writing to this bit are:

- If EL2 is implemented and enabled in the current Security state, in EL0 and EL1, if PMN is not 0, a write of 1 to this bit resets event counters in the range [0 .. (PMN-1)].
- If EL2 is disabled in the current Security state, a write of 1 to this bit resets all the event counters.
- In EL2 and EL3, a write of 1 to this bit resets all the event counters.
- This field does not affect the operation of other event counters and [PMCCNTR](#).

Note

Resetting the event counters does not change the event counter overflow bits.

If FEAT_PMUv3p5 is implemented, the values of [HDCR](#).HLP and PMCR.LP are ignored and bits [63:0] of all affected event counters are reset.

Access to this field is **WO/RAZ**.

E, bit [0]

Enable.

In the description of this field:

- If EL2 is implemented and is using AArch32, PMN is [HDCR](#).HPMN.
- If EL2 is implemented and is using AArch64, PMN is [MDCR_EL2](#).HPMN.
- If EL2 is not implemented, PMN is PMCR.N.

| E | Meaning |
|----------|--|
| 0b0 | PMCCNTR is disabled and event counters PMEVCNTR<n> , where n is in the range of affected event counters, are disabled. |
| 0b1 | PMCCNTR and event counters PMEVCNTR<n> where n is in the range of affected event counters, are enabled by PMCNTENSET . |

If PMN is not 0, this field affects the operation of event counters in the range [0 .. (PMN-1)].

This field does not affect the operation of other event counters.

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing PMCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPMCR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCR;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPMCR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMCR;

```

```
elsif PSTATE.EL == EL3 then  
    return PMCR;
```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMCR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPMCR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCR = R[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPMCR == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPMCR == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCR = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMCR = R[t];

```

```
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);  
    else  
        PMCR = R[t];  
    elsif PSTATE.EL == EL3 then  
        PMCR = R[t];
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMEVCNTR<n>, Performance Monitors Event Count Registers, n = 0 - 30

The PMEVCNTR<n> characteristics are:

Purpose

Holds event counter n, which counts events, where n is 0 to 30.

Configuration

AArch32 System register PMEVCNTR<n> bits [31:0] are architecturally mapped to AArch64 System register [PMEVCNTR<n>_EL0\[31:0\]](#).

AArch32 System register PMEVCNTR<n> bits [31:0] are architecturally mapped to External register [PMEVCNTR<n>_EL0\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMEVCNTR<n> are UNDEFINED.

Attributes

PMEVCNTR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Event counter n | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

If FEAT_PMUv3p5 is implemented, the event counter is 64 bits and only the least-significant part of the event counter is accessible in AArch32 state:

- Reads from PMEVCNTR<n> return bits [31:0] of the counter.
- Writes to PMEVCNTR<n> update bits [31:0] and leave bits [63:32] unchanged.
- There is no means to access bits [63:32] directly from AArch32 state.
- If the implementation does not support AArch64, bits [63:32] are not required to be implemented.

If FEAT_PMUv3p5 is not implemented, the event counter is 32 bits.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMEVCNTR<n>

PMEVCNTR<n> can also be accessed by using [PMXEVCNTR](#) with [PMSEL](#).SEL set to n.

If FEAT_FGT is implemented and <n> is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of [PMEVCNTR<n>](#) is as follows:

- If <n> is an unimplemented event counter, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and <n> is greater than or equal to the number of accessible event counters, then reads and writes of [PMEVCNTR<n>](#) are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- Accesses to the register behave as if <n> is an UNKNOWN value less-than-or-equal-to the index of the highest accessible event counter.
- If EL2 is implemented and enabled in the current Security state, and <n> is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Note

In EL0, an access is permitted if it is enabled by [PMUSERENR](#).{ER,EN} or [PMUSERENR_EL0](#).{ER,EN}.

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, [HDCR](#).HPMN identifies the number of accessible event counters.
- If EL2 is using AArch64, [MDCR_EL2](#).HPMN identifies the number of accessible event counters.

Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see [HDCR](#).HPMN and [MDCR_EL2](#).HPMN.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|-------------|--------|
| 0b1111 | 0b000 | 0b1110 | 0b10:n[4:3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elseif !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)];
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)];
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)];
elseif PSTATE.EL == EL3 then
    return PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|-------------|--------|
| 0b1111 | 0b000 | 0b1110 | 0b10:n[4:3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elseif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elseif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDBGWTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
elseif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
elseif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
elseif PSTATE.EL == EL3 then
    PMEVCNTR[UInt(CRm<1:0>:opc2<2:0>)] = R[t];

```

PMEVTYPER<n>, Performance Monitors Event Type Registers, n = 0 - 30

The PMEVTYPER<n> characteristics are:

Purpose

Configures event counter n, where n is 0 to 30.

Configuration

AArch32 System register PMEVTYPER<n> bits [31:0] are architecturally mapped to AArch64 System register [PMEVTYPER<n>_EL0\[31:0\]](#).

AArch32 System register PMEVTYPER<n> bits [31:0] are architecturally mapped to External register [PMEVTYPER<n>_EL0\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMEVTYPER<n> are UNDEFINED.

Attributes

PMEVTYPER<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|-----|-----|-----|------|----|------|-----|------|----|----|-----------------|----|----|---------------|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P | U | NSK | NSU | NSH | RES0 | MT | RES0 | RLU | RES0 | | | evtCount[15:10] | | | evtCount[9:0] | | | | | | | | | | | | | | | | |

P, bit [31]

Privileged filtering bit. Controls counting in EL1.

If EL3 is implemented, then counting in Non-secure EL1 is further controlled by the PMEVTYPER<n>.NSK bit.

| P | Meaning |
|-----|-----------------------------|
| 0b0 | Count events in EL1. |
| 0b1 | Do not count events in EL1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

U, bit [30]

User filtering bit. Controls counting in EL0.

If EL3 is implemented, then counting in Non-secure EL0 is further controlled by the PMEVTYPER<n>.NSU bit.

If FEAT_RME is implemented, then counting in Realm EL0 is further controlled by the PMEVTYPER<n>.RLU bit.

| U | Meaning |
|-----|-----------------------------|
| 0b0 | Count events in EL0. |
| 0b1 | Do not count events in EL0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 (kernel) modes filtering bit. Controls counting in Non-secure EL1.

If the value of this bit is equal to the value of PMEVTYPER<n>.P, events in Non-secure EL1 are counted.

Otherwise, events in Non-secure EL1 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When EL3 is implemented:

Non-secure EL0 (Unprivileged) filtering. Controls counting in Non-secure EL0.

If the value of this bit is equal to the value of PMEVTYPER<n>.U, events in Non-secure EL0 are counted.

Otherwise, events in Non-secure EL0 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When EL2 is implemented:

EL2 (Hyp mode) filtering bit. Controls counting in EL2.

| NSH | Meaning |
|-----|-----------------------------|
| 0b0 | Do not count events in EL2. |
| 0b1 | Count events in EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [26]

Reserved, RES0.

MT, bit [25]

When FEAT_MTPMU is implemented or an IMPLEMENTATION DEFINED multi-threaded PMU extension is implemented:

Multithreading.

| MT | Meaning |
|-----|--|
| 0b0 | Count events only on controlling PE. |
| 0b1 | Count events from any PE with the same affinity at level 1 and above as this PE. |

From Armv8.6, the IMPLEMENTATION DEFINED multi-threaded PMU extension is not permitted, meaning if FEAT_MTPMU is not implemented, this bit is RES0. See [ID_DFR1](#).MTPMU.

This bit is ignored by the PE and treated as zero when FEAT_MTPMU is implemented and Disabled.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [24:22]

Reserved, RES0.

RLU, bit [21]

When FEAT_RME is implemented:

Realm EL0 (unprivileged) filtering bit. Controls counting in Realm EL0.

If the value of this bit is equal to the value of the PMEVTYPER<n>.U bit, events in Realm EL0 are counted.

Otherwise, events in Realm EL0 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [20:16]

Reserved, RES0.

evtCount[15:10], bits [15:10]

When FEAT_PMUv3p1 is implemented:

Extension to evtCount[9:0]. For more information, see evtCount[9:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

evtCount[9:0], bits [9:0]

Event to count.

The event number of the event that is counted by event counter [PMEVCNTR<n>](#).

The ranges of event numbers allocated to each type of event are shown in 'Allocation of the PMU event number space'.

If FEAT_PMUv3p8 is implemented and PMEVTYPER<n>.evtCount is programmed to an event that is reserved or not supported by the PE, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>.evtCount field is the value written to the field.

Otherwise, if PMEVTYPER<n>.evtCount is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range 0x0000 to 0x003F, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>.evtCount field is the value written to the field.
- If FEAT_PMUv3p1 is implemented, for the range 0x4000 to 0x403F, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>.evtCount field is the value written to the field.
- For other values, it is UNPREDICTABLE what event, if any, is counted and the value returned by a direct or external read of the PMEVTYPER<n>.evtCount field is UNKNOWN.

Note

UNPREDICTABLE means the event must not expose privileged information.

Arm recommends that for all values that represent reserved or unsupported events, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>.evtCount field is the value written to the field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMEVTYPER<n>

PMEVTYPER<n> can also be accessed by using [PMXEVTYPER](#) with [PMSCLR](#).SEL set to n.

If FEAT_FGT is implemented and <n> is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of [PMEVTYPER<n>](#) is as follows:

- If <n> is an unimplemented event counter, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and <n> is greater than or equal to the number of accessible event counters, then reads and writes of [PMEVTYPER<n>](#) are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP.
- Accesses to the register behave as if <n> is an UNKNOWN value less-than-or-equal-to the index of the highest accessible event counter.
- If EL2 is implemented and enabled in the current Security state, and <n> is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Note

In EL0, an access is permitted if it is enabled by [PMUSERENR](#).EN or [PMUSERENR_EL0](#).EN.

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, [HDCR](#).HPMN identifies the number of accessible event counters.
- If EL2 is using AArch64, [MDCR_EL2](#).HPMN identifies the number of accessible event counters.

Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see [HDCR](#).HPMN and [MDCR_EL2](#).HPMN.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|-------------|--------|
| 0b1111 | 0b000 | 0b1110 | 0b11:n[4:3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVTYPEPERn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)];
elsif PSTATE.EL == EL3 then
    return PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)];

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|-------------|--------|
| 0b1111 | 0b000 | 0b1110 | 0b11:n[4:3] | n[2:0] |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTen == '1') && HDFGWTR_EL2.PMEVTYPEPERn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)] = R[t];
elsif PSTATE.EL == EL3 then
    PMEVTYPER[UInt(CRm<1:0>:opc2<2:0>)] = R[t];

```


PMINTENCLR, Performance Monitors Interrupt Enable Clear register

The PMINTENCLR characteristics are:

Purpose

Disables the generation of interrupt requests on overflows from the Cycle Count Register, [PMCCNTR](#), and the event counters [PMEVCNTR<n>](#). Reading the register shows which overflow interrupt requests are enabled.

PMINTENCLR is used in conjunction with the [PMINTENSET](#) register.

Configuration

AArch32 System register PMINTENCLR bits [31:0] are architecturally mapped to AArch64 System register [PMINTENCLR_EL1\[31:0\]](#).

AArch32 System register PMINTENCLR bits [31:0] are architecturally mapped to External register [PMINTENCLR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMINTENCLR are UNDEFINED.

Attributes

PMINTENCLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

C, bit [31]

[PMCCNTR](#) overflow interrupt request disable bit.

| C | Meaning |
|-----|--|
| 0b0 | When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect. |
| 0b1 | When read, means the cycle counter overflow interrupt request is enabled. When written, disables the cycle count overflow interrupt request. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter overflow interrupt request disable bit for [PMEVCNTR<n>](#).

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1, N is the value in [MDCR_EL2](#).HPMN if EL2 is using AArch64, or in [HDCR](#).HPMN if EL2 is using AArch32. Otherwise, N is the value in [PMCR](#).N.

| P<n> | Meaning |
|------|---|
| 0b0 | When read, means that the PMEVCNTR<n> event counter interrupt request is disabled. When written, has no effect. |
| 0b1 | When read, means that the PMEVCNTR<n> event counter interrupt request is enabled. When written, disables the PMEVCNTR<n> interrupt request. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMINTENCLR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return PMINTENCLR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            else
                return PMINTENCLR;
    elsif PSTATE.EL == EL3 then
        return PMINTENCLR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            PMINTENCLR = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            else
                PMINTENCLR = R[t];
    elsif PSTATE.EL == EL3 then
        PMINTENCLR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMINTENSET, Performance Monitors Interrupt Enable Set register

The PMINTENSET characteristics are:

Purpose

Enables the generation of interrupt requests on overflows from the Cycle Count Register, [PMCCNTR](#), and the event counters [PMEVCNTR<n>](#). Reading the register shows which overflow interrupt requests are enabled.

PMINTENSET is used in conjunction with the [PMINTENCLR](#) register.

Configuration

AArch32 System register PMINTENSET bits [31:0] are architecturally mapped to AArch64 System register [PMINTENSET_EL1\[31:0\]](#).

AArch32 System register PMINTENSET bits [31:0] are architecturally mapped to External register [PMINTENSET_EL1\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMINTENSET are UNDEFINED.

Attributes

PMINTENSET is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

C, bit [31]

[PMCCNTR](#) overflow interrupt request enable bit.

| C | Meaning |
|-----|---|
| 0b0 | When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect. |
| 0b1 | When read, means the cycle counter overflow interrupt request is enabled. When written, enables the cycle count overflow interrupt request. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter overflow interrupt request enable bit for [PMEVCNTR<n>](#).

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1, N is the value in [MDCR_EL2](#).HPMN if EL2 is using AArch64, or in [HDCR](#).HPMN if EL2 is using AArch32. Otherwise, N is the value in [PMCR](#).N.

| P<n> | Meaning |
|------|--|
| 0b0 | When read, means that the PMEVCNTR<n> event counter interrupt request is disabled. When written, has no effect. |
| 0b1 | When read, means that the PMEVCNTR<n> event counter interrupt request is enabled. When written, enables the PMEVCNTR<n> interrupt request. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMINTENSET

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return PMINTENSET;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            else
                return PMINTENSET;
    elsif PSTATE.EL == EL3 then
        return PMINTENSET;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            PMINTENSET = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            else
                PMINTENSET = R[t];
    elsif PSTATE.EL == EL3 then
        PMINTENSET = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMMIR, Performance Monitors Machine Identification Register

The PMMIR characteristics are:

Purpose

Describes Performance Monitors parameters specific to the implementation to software.

Configuration

This register is present only when AArch32 is supported and FEAT_PMUv3p4 is implemented. Otherwise, direct accesses to PMMIR are UNDEFINED.

Attributes

PMMIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|---------|----|----|----|-----------|----|----|----|-----------|----|----|----|----|----|-------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | THWIDTH | | | | BUS_WIDTH | | | | BUS_SLOTS | | | | | | SLOTS | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

THWIDTH, bits [23:20]

[PMEVTYPER<n>_EL0](#).TH width. Indicates implementation of the FEAT_PMUv3_TH feature, and, if implemented, the size of the [PMEVTYPER<n>_EL0](#).TH field.

| THWIDTH | Meaning |
|---------|--|
| 0b0000 | FEAT_PMUv3_TH is not implemented. |
| 0b0001 | 1 bit. PMEVTYPER<n>_EL0 .TH[11:1] are RES0. |
| 0b0010 | 2 bits. PMEVTYPER<n>_EL0 .TH[11:2] are RES0. |
| 0b0011 | 3 bits. PMEVTYPER<n>_EL0 .TH[11:3] are RES0. |
| 0b0100 | 4 bits. PMEVTYPER<n>_EL0 .TH[11:4] are RES0. |
| 0b0101 | 5 bits. PMEVTYPER<n>_EL0 .TH[11:5] are RES0. |
| 0b0110 | 6 bits. PMEVTYPER<n>_EL0 .TH[11:6] are RES0. |
| 0b0111 | 7 bits. PMEVTYPER<n>_EL0 .TH[11:7] are RES0. |
| 0b1000 | 8 bits. PMEVTYPER<n>_EL0 .TH[11:8] are RES0. |
| 0b1001 | 9 bits. PMEVTYPER<n>_EL0 .TH[11:9] are RES0. |
| 0b1010 | 10 bits. PMEVTYPER<n>_EL0 .TH[11:10] are RES0. |
| 0b1011 | 11 bits. PMEVTYPER<n>_EL0 .TH[11] is RES0. |
| 0b1100 | 12 bits. |

All other values are reserved.

If FEAT_PMUv3_TH is not implemented, this field is zero.

Otherwise, the largest value that can be written to [PMEVTYPER<n>_EL0](#).TH is $2^{(\text{PMMIR.THWIDTH})}$ minus one.

Note

[PMEVTYPER<n>_EL0](#).TH cannot be accessed through [PMEVTYPER<n>](#).

Access to this field is **RO**.

BUS_WIDTH, bits [19:16]

Bus width. Indicates the number of bytes each BUS_ACCESS event relates to. Encoded as $\text{Log}_2(\text{number of bytes})$, plus one. Defined values are:

| BUS_WIDTH | Meaning |
|------------------|-----------------------------------|
| 0b0000 | The information is not available. |
| 0b0011 | Four bytes. |
| 0b0100 | 8 bytes. |
| 0b0101 | 16 bytes. |
| 0b0110 | 32 bytes. |
| 0b0111 | 64 bytes. |
| 0b1000 | 128 bytes. |
| 0b1001 | 256 bytes. |
| 0b1010 | 512 bytes. |
| 0b1011 | 1024 bytes. |
| 0b1100 | 2048 bytes. |

All other values are reserved.

Each transfer is up to this number of bytes. An access might be smaller than the bus width.

When this field is nonzero, each access counted by BUS_ACCESS is at most BUS_WIDTH bytes. An implementation might treat a wide bus as multiple narrower buses, such that a wide access on the bus increments the BUS_ACCESS counter by more than one.

Access to this field is **RO**.

BUS_SLOTS, bits [15:8]

Bus count. The largest value by which the BUS_ACCESS event might increment by in a single BUS_CYCLES cycle.

When this field is nonzero, the largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle is BUS_SLOTS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SLOTS, bits [7:0]

Operation width. The largest value by which the STALL_SLOT event might increment by in a single cycle. If the STALL_SLOT event is not implemented, this field might read as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMMIR

Accesses to this register use the following encodings in the System register encoding space:

$\text{MRC}\{\langle c \rangle\}\{\langle q \rangle\} \langle \text{coproc} \rangle, \{ \# \} \langle \text{opc1} \rangle, \langle \text{Rt} \rangle, \langle \text{CRn} \rangle, \langle \text{CRm} \rangle \{, \{ \# \} \langle \text{opc2} \rangle \}$

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return PMMIR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            else
                return PMMIR;
    elsif PSTATE.EL == EL3 then
        return PMMIR;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMOVSr, Performance Monitors Overflow Flag Status Register

The PMOVSr characteristics are:

Purpose

Contains the state of the overflow bit for the Cycle Count Register, [PMCCNTR](#), and each of the implemented event counters [PMEVCNTR<n>](#). Writing to this register clears these bits.

Configuration

AArch32 System register PMOVSr bits [31:0] are architecturally mapped to AArch64 System register [PMOVSCLR_EL0\[31:0\]](#).

AArch32 System register PMOVSr bits [31:0] are architecturally mapped to External register [PMOVSCLR_EL0\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMuV3 is implemented. Otherwise, direct accesses to PMOVSr are UNDEFINED.

Attributes

PMOVSr is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

C, bit [31]

Cycle counter overflow clear bit. Possible values are:

| C | Meaning |
|-----|--|
| 0b0 | When read, means the cycle counter has not overflowed since this bit was last cleared. When written, has no effect. |
| 0b1 | When read, means the cycle counter has overflowed since this bit was last cleared. When written, clears the cycle counter overflow bit to 0. |

[PMCR](#).LC controls whether an overflow is detected from unsigned overflow of [PMCCNTR](#)[31:0] or unsigned overflow of [PMCCNTR](#)[63:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter overflow clear bit for [PMEVCNTR<n>](#).

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in [MDCR_EL2](#).HPMN if EL2 is using AArch64, or in [HDCR](#).HPMN if EL2 is using AArch32. Otherwise, N is the value in [PMCR](#).N.

| P<n> | Meaning |
|-------------------|---|
| 0b0 | When read, means that PMEVCNTR<n> has not overflowed since this bit was last cleared. When written, has no effect. |
| 0b1 | When read, means that PMEVCNTR<n> has overflowed since this bit was last cleared. When written, clears the PMEVCNTR<n> overflow bit to 0. |

If FEAT_PMUv3p5 is implemented, [MDCR_EL2.HLP](#), [HDCR.HLP](#), and [PMCR.LP](#) control whether an overflow is detected from unsigned overflow of [PMEVCNTR<n>](#)[31:0] or unsigned overflow of [PMEVCNTR<n>](#)[63:0]. [PMEVCNTR<n>](#)[63:32] cannot be accessed directly in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMOVSr

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b011 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMOVS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMOVSR;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMOVSR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMOVSR;
elsif PSTATE.EL == EL3 then
    return PMOVSR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b011 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMOVS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMOVSr = R[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMOVSr = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMOVSr = R[t];
elsif PSTATE.EL == EL3 then
    PMOVSr = R[t];

```


PMOVSSET, Performance Monitors Overflow Flag Status Set register

The PMOVSSET characteristics are:

Purpose

Sets the state of the overflow bit for the Cycle Count Register, [PMCCNTR](#), and each of the implemented event counters [PMEVCNTR<n>](#).

Configuration

AArch32 System register PMOVSSET bits [31:0] are architecturally mapped to AArch64 System register [PMOVSSET_EL0\[31:0\]](#).

AArch32 System register PMOVSSET bits [31:0] are architecturally mapped to External register [PMOVSSET_EL0\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMOVSSET are UNDEFINED.

Attributes

PMOVSSET is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

C, bit [31]

Cycle counter overflow set bit.

| C | Meaning |
|-----|--|
| 0b0 | When read, means the cycle counter has not overflowed since this bit was last cleared. When written, has no effect. |
| 0b1 | When read, means the cycle counter has overflowed since this bit was last cleared. When written, sets the cycle counter overflow bit to 1. |

[PMCR.LC](#) controls whether an overflow is detected from unsigned overflow of [PMCCNTR](#)[31:0] or unsigned overflow of [PMCCNTR](#)[63:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter overflow set bit for [PMEVCNTR<n>](#).

If N is less than 31, then bits [30:N] are RAZ/WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in [MDCR_EL2.HPMN](#) if EL2 is using AArch64, or in [HDCR.HPMN](#) if EL2 is using AArch32. Otherwise, N is the value in [PMCR.N](#).

| P<n> | Meaning |
|-------------------|--|
| 0b0 | When read, means that PMEVCNTR<n> has not overflowed since this bit was last cleared. When written, has no effect. |
| 0b1 | When read, means that PMEVCNTR<n> has overflowed since this bit was last . When written, sets the PMEVCNTR<n> overflow bit to 1. |

If FEAT_PMUv3p5 is implemented, [MDCR_EL2.HLP](#), [HDCR.HLP](#), and [PMCR.LP](#) control whether an overflow is detected from unsigned overflow of [PMEVCNTR<n>](#)[31:0] or unsigned overflow of [PMEVCNTR<n>](#)[63:0]. [PMEVCNTR<n>](#)[63:32] cannot be accessed directly in AArch32 state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMOVSSET

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b011 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMOVS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMOVSSSET;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMOVSSSET;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMOVSSSET;
elsif PSTATE.EL == EL3 then
    return PMOVSSSET;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b011 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMOVS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMOVSSSET = R[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMOVSSSET = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMOVSSSET = R[t];
elsif PSTATE.EL == EL3 then
    PMOVSSSET = R[t];

```


PMSELR, Performance Monitors Event Counter Selection Register

The PMSELR characteristics are:

Purpose

Selects the current event counter [PMEVCNTR<n>](#) or the cycle counter, CCNT.

PMSELR is used in conjunction with [PMXEVTYPER](#) to determine the event that increments a selected event counter, and the modes and states in which the selected counter increments.

It is also used in conjunction with [PMXEVNTR](#), to determine the value of a selected event counter.

Configuration

AArch32 System register PMSELR bits [31:0] are architecturally mapped to AArch64 System register [PMSELR_EL0\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMuV3 is implemented. Otherwise, direct accesses to PMSELR are UNDEFINED.

Attributes

PMSELR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | SEL | | | | | | | | | | | | | |

Bits [31:5]

Reserved, RES0.

SEL, bits [4:0]

Selects event counter, [PMEVCNTR<n>](#), where n is the value held in this field. This value identifies which event counter is accessed when a subsequent access to [PMXEVTYPER](#) or [PMXEVNTR](#) occurs.

This field can take any value from 0 (0b00000) to (PMCR.N)-1, or 31 (0b11111).

When PMSELR.SEL is 0b11111, it selects the cycle counter and:

- A read of the [PMXEVTYPER](#) returns the value of [PMCCFILTR](#).
- A write of the [PMXEVTYPER](#) writes to [PMCCFILTR](#).
- A read or write of [PMXEVNTR](#) has CONSTRAINED UNPREDICTABLE effects. For more information, see [PMXEVNTR](#).

For more information about the results of accesses to event counters, see [PMXEVTYPER](#) and [PMXEVNTR](#).

For more information about the number of counters accessible at each Exception level, see [HDCR.HPMN](#) and [MDCR_EL2.HPMN](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMSELR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b101 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMSELR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMSELR;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMSELR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMSELR;
elsif PSTATE.EL == EL3 then
    return PMSELR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b101 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMSELR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMSELR = R[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMSELR = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMSELR = R[t];
elsif PSTATE.EL == EL3 then
    PMSELR = R[t];

```


PMSWINC, Performance Monitors Software Increment register

The PMSWINC characteristics are:

Purpose

Increments a counter that is configured to count the Software increment event, event 0x00. For more information, see 'SW_INCR'.

Configuration

AArch32 System register PMSWINC bits [31:0] are architecturally mapped to AArch64 System register [PMSWINC_EL0\[31:0\]](#).

AArch32 System register PMSWINC bits [31:0] are architecturally mapped to External register [PMSWINC_EL0\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMSWINC are UNDEFINED.

Attributes

PMSWINC is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

Bit [31]

Reserved, RES0.

P<n>, bit [n], for n = 30 to 0

Event counter software increment bit for [PMEVCNTR<n>](#).

If N is less than 31, then bits [30:N] are WI. When EL2 is implemented and enabled in the current Security state, in EL1 and EL0, N is the value in [MDCR_EL2](#).HPMN if EL2 is using AArch64, or in [HDCR](#).HPMN if EL2 is using AArch32. Otherwise, N is the value in [PMCR](#).N.

| P<n> | Meaning |
|------|--|
| 0b0 | No action. The write to this bit is ignored. |
| 0b1 | If PMEVCNTR<n> is enabled and configured to count the software increment event, increments PMEVCNTR<n> by 1. If PMEVCNTR<n> is disabled, or not configured to count the software increment event, the write to this bit is ignored. |

Accessing PMSWINC

Accesses to this register use the following encodings in the System register encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| | | | | |
|--------|------|-----|-----|------|
| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b100 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.<SW,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
        elsif ELUsingAArch32(EL1) && PMUSERENR.<SW,EN> == '00' then
            if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
                AArch64.AArch32SystemAccessTrap(EL2, 0x03);
            elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
                AArch32.TakeHypTrapException(0x00);
            else
                UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEN == '1') && HDFGWTR_EL2.PMSWINC_EL0 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            PMSWINC = R[t];
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            PMSWINC = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            PMSWINC = R[t];
    elsif PSTATE.EL == EL3 then
        PMSWINC = R[t];

```


PMUSERENR, Performance Monitors User Enable Register

The PMUSERENR characteristics are:

Purpose

Enables or disables User mode access to the Performance Monitors.

Configuration

AArch32 System register PMUSERENR bits [31:0] are architecturally mapped to AArch64 System register [PMUSERENR_EL0\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMUSERENR are UNDEFINED.

Attributes

PMUSERENR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | ER | | | | CR | SW | EN | | | | | | | |

Bits [31:4]

Reserved, RES0.

ER, bit [3]

Event counter read trap control:

| ER | Meaning |
|-----|--|
| 0b0 | EL0 reads of the PMXVCNTR and PMEVCNTR<n> , and EL0 RW access to the PMSELR , are trapped to Undefined mode if PMUSERENR.EN is also 0. |
| 0b1 | Overrides PMUSERENR.EN and enables RO access to PMXVCNTR and PMEVCNTR<n> , and RW access to PMSELR . |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

CR, bit [2]

Cycle counter read trap control:

| CR | Meaning |
|-----|---|
| 0b0 | EL0 reads of the PMCCNTR are trapped to Undefined mode if PMUSERENR.EN is also 0. |
| 0b1 | Overrides PMUSERENR.EN and enables access to PMCCNTR . |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

SW, bit [1]

Software increment write trap control:

| SW | Meaning |
|-----|--|
| 0b0 | EL0 writes to the PMSWINC are trapped to Undefined mode if PMUSERENR.EN is also 0. |
| 0b1 | Overrides PMUSERENR.EN and enables access to PMSWINC . |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

EN, bit [0]

Traps EL0 accesses to the Performance Monitors registers to Undefined mode, as follows:

- [PMCR](#), [PMOVSr](#), [PMSELR](#), [PMCEID0](#), [PMCEID1](#), [PMCCNTR](#), [PMXEVTYPER](#), [PMXVCNTR](#), [PMCNTENSET](#), [PMCNTENCLR](#), [PMOVSSET](#), [PMEVCNTR<n>](#), [PMEVTYPER<n>](#), [PMCCFILTR](#), [PMSWINC](#).
- If FEAT_PMUv3p1 is implemented, [PMCEID2](#), and [PMCEID3](#).
- If FEAT_PMUv3p4 is implemented, [PMMIR](#).

| EN | Meaning |
|-----|---|
| 0b0 | While at EL0, accesses to the specified registers at EL0 are trapped to Undefined mode, unless overridden by one of PMUSERENR.{ER, CR, SW}. |
| 0b1 | While at EL0, software can access all of the specified registers. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing PMUSERENR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b000 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMUSERENR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            return PMUSERENR;
    elsif PSTATE.EL == EL1 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
            AArch32.TakeHypTrapException(0x03);
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            else
                return PMUSERENR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            else
                return PMUSERENR;
    elsif PSTATE.EL == EL3 then
        return PMUSERENR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        else
            PMUSERENR = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
            else
                PMUSERENR = R[t];
    elsif PSTATE.EL == EL3 then
        PMUSERENR = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMXEVNTR, Performance Monitors Selected Event Count Register

The PMXEVNTR characteristics are:

Purpose

Reads or writes the value of the selected event counter, [PMEVCNTR<n>](#). [PMSELR](#).SEL determines which event counter is selected.

Configuration

AArch32 System register PMXEVNTR bits [31:0] are architecturally mapped to AArch64 System register [PMXEVNTR_EL0\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMXEVNTR are UNDEFINED.

Attributes

PMXEVNTR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| PMEVCNTR<n> | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

PMEVCNTR<n>, bits [31:0]

Value of the selected event counter, [PMEVCNTR<n>](#), where n is the value stored in [PMSELR](#).SEL.

If FEAT_PMUv3p5 is implemented, the event counter is 64 bits and only the least-significant part of the event counter is accessible in AArch32 state:

- Reads from PMXEVNTR return bits [31:0] of the counter.
- Writes to PMXEVNTR update bits [31:0] and leave bits [63:32] unchanged.
- There is no means to access bits [63:32] directly from AArch32 state.
- If the implementation does not support AArch64, bits [63:32] are not required to be implemented.

If FEAT_PMUv3p5 is not implemented, the event counter is 32 bits.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMXEVNTR

If FEAT_FGT is implemented and [PMSELR](#).SEL is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of [PMXEVNTR](#) is as follows:

- If [PMSELR](#).SEL selects an unimplemented event counter, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented and [PMSELR](#).SEL is greater than or equal to the number of accessible event counters, then reads and writes of [PMXEVNTR](#) are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP
- Accesses to the register behave as if [PMSELR.SEL](#) has an UNKNOWN value less than the number of event counters accessible at the current Exception level and Security state.
- If EL2 is implemented and enabled in the current Security state, and [PMSELR.SEL](#) is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Note

In EL0, an access is permitted if it is enabled by [PMUSERENR.{ER,EN}](#) or [PMUSERENR_EL0.{ER,EN}](#).

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, [HDCR.HPMN](#) identifies the number of accessible event counters.
- If EL2 is using AArch64, [MDCR_EL2.HPMN](#) identifies the number of accessible event counters.

Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see [HDCR.HPMN](#) and [MDCR_EL2.HPMN](#).

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1101 | 0b010 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR.<ER,EN> == '00' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMXEVCNTR;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMXEVCNTR;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMXEVCNTR;
elsif PSTATE.EL == EL3 then
    return PMXEVCNTR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1101 | 0b010 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVCNTRn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMXEVCNTR = R[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMXEVCNTR = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMXEVCNTR = R[t];
elsif PSTATE.EL == EL3 then
    PMXEVCNTR = R[t];

```


PMXEVTYPER, Performance Monitors Selected Event Type Register

The PMXEVTYPER characteristics are:

Purpose

When [PMSELR.SEL](#) selects an event counter, this accesses a [PMEVTYPER<n>](#) register. When [PMSELR.SEL](#) selects the cycle counter, this accesses [PMCCFILTR](#).

Configuration

AArch32 System register PMXEVTYPER bits [31:0] are architecturally mapped to AArch64 System register [PMXEVTYPER_ELO\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_PMUv3 is implemented. Otherwise, direct accesses to PMXEVTYPER are UNDEFINED.

Attributes

PMXEVTYPER is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Event type register or PMCCFILTR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Event type register or [PMCCFILTR](#).

When [PMSELR.SEL](#) == 31, this register accesses [PMCCFILTR](#).

Otherwise, this register accesses [PMEVTYPER<n>](#) where n is the value in [PMSELR.SEL](#).

Accessing PMXEVTYPER

If FEAT_FGT is implemented, and [PMSELR.SEL](#) is not 31 and is greater than or equal to the number of accessible event counters, then the behavior of permitted reads and writes of [PMXEVTYPER](#) is as follows:

- If [PMSELR.SEL](#) selects an unimplemented event counter, the access is UNDEFINED.
- Otherwise, the access is trapped to EL2.

If FEAT_FGT is not implemented, and [PMSELR.SEL](#) is not 31 and is greater than or equal to the number of accessible event counters, then reads and writes of [PMXEVTYPER](#) are CONSTRAINED UNPREDICTABLE, and the following behaviors are permitted:

- Accesses to the register are UNDEFINED.
- Accesses to the register behave as RAZ/WI.
- Accesses to the register execute as a NOP
- Accesses to the register behave as if [PMSELR.SEL](#) has an UNKNOWN value less than the number of event counters accessible at the current Exception level and Security state.
- Accesses to the register behave as if [PMSELR.SEL](#) is 31.
- If EL2 is implemented and enabled in the current Security state, and [PMSELR.SEL](#) is less than the number of implemented event counters, accesses from EL1 or permitted accesses from EL0 are trapped to EL2.

Note

In EL0, an access is permitted if it is enabled by [PMUSERENR](#).EN or [PMUSERENR_EL0](#).EN.

If EL2 is implemented and enabled in the current Security state, at EL0 and EL1:

- If EL2 is using AArch32, [HDCR](#).HPMN identifies the number of accessible event counters.
- If EL2 is using AArch64, [MDCR_EL2](#).HPMN identifies the number of accessible event counters.

Otherwise, the number of accessible event counters is the number of implemented event counters. For more information, see [HDCR](#).HPMN and [MDCR_EL2](#).HPMN.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1001 | 0b1101 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGRTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMXEVTYPER;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMXEVTYPER;
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return PMXEVTYPER;
elsif PSTATE.EL == EL3 then
    return PMXEVTYPER;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1001 | 0b1101 | 0b001 |

```

if PSTATE.EL == EL0 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif !ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        else
            AArch64.AArch32SystemAccessTrap(EL1, 0x03);
    elsif ELUsingAArch32(EL1) && PMUSERENR_EL0.EN == '0' then
        if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TGE == '1' then
            AArch64.AArch32SystemAccessTrap(EL2, 0x03);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TGE == '1' then
            AArch32.TakeHypTrapException(0x00);
        else
            UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T9 == '1'
then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HDFGWTR_EL2.PMEVTYPERn_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMXEVTYPER = R[t];
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T9 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T9 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TPM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TPM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMXEVTYPER = R[t];
elsif PSTATE.EL == EL2 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        UNDEFINED;
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TPM == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        PMXEVTYPER = R[t];
elsif PSTATE.EL == EL3 then
    PMXEVTYPER = R[t];

```


PRRR, Primary Region Remap Register

The PRRR characteristics are:

Purpose

Controls the top level mapping of the TEX[0], C, and B memory region attributes.

Configuration

AArch32 System register PRRR bits [31:0] are architecturally mapped to AArch64 System register [MAIR_EL1\[31:0\]](#) when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register PRRR bits [31:0] are architecturally mapped to AArch32 System register [MAIR0\[31:0\]](#) when EL3 is not implemented or EL3 is using AArch64.

AArch32 System register PRRR bits [31:0] (PRRR_S) are architecturally mapped to AArch32 System register [MAIR0\[31:0\]](#) (MAIR0_S) when EL3 is using AArch32.

AArch32 System register PRRR bits [31:0] (PRRR_NS) are architecturally mapped to AArch32 System register [MAIR0\[31:0\]](#) (MAIR0_NS) when EL3 is using AArch32.

This register is present only when AArch32 is supported. Otherwise, direct accesses to PRRR are UNDEFINED.

[MAIR0](#) and PRRR are the same register, with a different view depending on the value of [TTBCR.EAE](#):

- When it is set to 0, the register is as described in PRRR.
- When it is set to 1, the register is as described in [MAIR0](#).

Attributes

PRRR is a 32-bit register.

Field descriptions

When TTBCR.EAE == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|------|------|------|------|------|------|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NOS7 | NOS6 | NOS5 | NOS4 | NOS3 | NOS2 | NOS1 | NOS0 | RES0 | NS1 | NS0 | DS1 | DS0 | TR7 | TR6 | TR5 | TR4 | TR3 | TR2 | TR1 | TR0 | | | | | | | | | | | |

NOS<n>, bit [n+24], for n = 7 to 0

Not Outer Shareable. NOS<n> is the Outer Shareable property for memory attributes n, if the region is mapped as Normal memory that is not Inner Non-cacheable, Outer Non-cacheable, and the appropriate PRRR.{NS0, NS1} field identifies the region as shareable. n is the value of the concatenation of the {TEX[0], C, B} bits from the translation table descriptor. The possible values of each NOS<n> field other than NOS6 are:

| NOS<n> | Meaning |
|--------|-----------------------------------|
| 0b0 | Memory region is Outer Shareable. |
| 0b1 | Memory region is Inner Shareable. |

The value of this bit is ignored if the region is:

- Device memory
- Normal memory that is at least one of:
 - Inner Non-cacheable, Outer Non-cacheable.
 - Identified by the appropriate PRRR.{NS0, NS1} field as Non-shareable.

The meaning of the NOS6 field is IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [23:20]

Reserved, RES0.

NS1, bit [19]

Mapping of S = 1 attribute for Normal memory regions. This field is used in determining the Shareability of a memory region that is mapped to Normal memory and both:

- Is not Inner Non-cacheable, Outer Non-cacheable.
- Has the S bit in the translation table descriptor set to 1.

| NS1 | Meaning |
|-----|--|
| 0b0 | Region is Non-shareable. |
| 0b1 | Region is shareable. The value of the appropriate PRRR.NOS<n> field determines whether the region is Inner Shareable or Outer Shareable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NS0, bit [18]

Mapping of S = 0 attribute for Normal memory regions. This field is used in determining the Shareability of a memory region that is mapped to Normal memory and both:

- Is not Inner Non-cacheable, Outer Non-cacheable.
- Has the S bit in the translation table descriptor set to 0.

| NS0 | Meaning |
|-----|--|
| 0b0 | Region is Non-shareable. |
| 0b1 | Region is shareable. The value of the appropriate PRRR.NOS<n> field determines whether the region is Inner Shareable or Outer Shareable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DS1, bit [17]

Mapping of S = 1 attribute for Device memory. From Armv8, all types of Device memory are Outer Shareable, and therefore this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

DS0, bit [16]

Mapping of S = 0 attribute for Device memory. From Armv8, all types of Device memory are Outer Shareable, and therefore this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

TR<n>, bits [2n+1:2n], for n = 7 to 0

TR<n> is the primary TEX mapping for memory attributes n, and defines the mapped memory type for a region with attributes n. n is the value of the concatenation of the {TEX[0], C, B} bits from the translation table descriptor. The possible values for each field other than TR6 are:

| TR<n> | Meaning |
|--------------------|----------------------|
| 0b00 | Device-nGnRnE memory |
| 0b01 | Device-nGnRE memory |
| 0b10 | Normal memory |

The value 0b11 is reserved. The effect of programming a field to 0b11 is CONSTRAINED UNPREDICTABLE.

The meaning of the TR6 field is IMPLEMENTATION DEFINED.

When FEAT_XS is implemented, stage 1 Inner Write-Back Cacheable, Outer Write-Back Cacheable memory types have the XS attribute set to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PRRR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|---------------|-------------|------------|------------|-------------|
| 0b1111 | 0b000 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            return MAIR0_NS;
        else
            return PRRR_NS;
    else
        if TTBCR.EAE == '1' then
            return MAIR0;
        else
            return PRRR;
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            if TTBCR.EAE == '1' then
                return MAIR0_NS;
            else
                return PRRR_NS;
        else
            if TTBCR.EAE == '1' then
                return MAIR0;
            else
                return PRRR;
    elsif PSTATE.EL == EL3 then
        if TTBCR.EAE == '1' then
            if SCR.NS == '0' then
                return MAIR0_S;
            else
                return MAIR0_NS;
        else
            if SCR.NS == '0' then
                return PRRR_S;
            else
                return PRRR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1010 | 0b0010 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T10 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T10 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        if TTBCR.EAE == '1' then
            MAIR0_NS = R[t];
        else
            PRRR_NS = R[t];
        else
            if TTBCR.EAE == '1' then
                MAIR0 = R[t];
            else
                PRRR = R[t];
    elsif PSTATE.EL == EL2 then
        if HaveEL(EL3) && ELUsingAArch32(EL3) then
            if TTBCR.EAE == '1' then
                MAIR0_NS = R[t];
            else
                PRRR_NS = R[t];
        else
            if TTBCR.EAE == '1' then
                MAIR0 = R[t];
            else
                PRRR = R[t];
    elsif PSTATE.EL == EL3 then
        if SCR.NS == '0' && CP15SDISABLE == HIGH then
            UNDEFINED;
        elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
            UNDEFINED;
        else
            if TTBCR.EAE == '1' then
                if SCR.NS == '0' then
                    MAIR0_S = R[t];
                else
                    MAIR0_NS = R[t];
            else
                if SCR.NS == '0' then
                    PRRR_S = R[t];
                else
                    PRRR_NS = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

REVIDR, Revision ID Register

The REVIDR characteristics are:

Purpose

Provides implementation-specific minor revision information.

Configuration

AArch32 System register REVIDR bits [31:0] are architecturally mapped to AArch64 System register [REVIDR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to REVIDR are UNDEFINED.

If REVIDR has the same value as [MIDR](#), then its contents have no significance.

Attributes

REVIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing REVIDR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return REVIDR;
elsif PSTATE.EL == EL2 then
    return REVIDR;
elsif PSTATE.EL == EL3 then
    return REVIDR;

```


RMR, Reset Management Register

The RMR characteristics are:

Purpose

If EL1 or EL3 is the highest implemented Exception level and this register is implemented:

- A write to the register at the highest implemented Exception level can request a Warm reset.
- If the highest implemented Exception level can use AArch32 and AArch64, this register specifies the Execution state that the PE boots into on a Warm reset.

Configuration

AArch32 System register RMR bits [31:0] are architecturally mapped to AArch64 System register [RMR_EL1\[31:0\]](#) when the highest implemented Exception level is EL1.

AArch32 System register RMR bits [31:0] are architecturally mapped to AArch64 System register [RMR_EL3\[31:0\]](#) when EL3 is implemented.

This register is present only when AArch32 is supported. Otherwise, direct accesses to RMR are UNDEFINED.

Only implemented if EL1 or EL3 is the highest implemented Exception level. In this case:

- If the highest implemented Exception level can use AArch32 and AArch64 then this register must be implemented.
- If the highest implemented Exception level cannot use AArch64 then it is IMPLEMENTATION DEFINED whether the register is implemented.

Attributes

RMR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | RR | | AA64 | | | | | | | | | | | | | |

Bits [31:2]

Reserved, RES0.

RR, bit [1]

Reset Request. Setting this bit to 1 requests a Warm reset.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

AA64, bit [0]

When the highest implemented Exception level can use AArch64, determines which Execution state the PE boots into after a Warm reset:

| AA64 | Meaning |
|------|----------|
| 0b0 | AArch32. |
| 0b1 | AArch64. |

On coming out of the Warm reset, execution starts at the IMPLEMENTATION DEFINED reset vector address of the specified Execution state.

If the highest implemented Exception level cannot use AArch64 this bit is RAZ/WI.

When implemented as a RW field, this field resets to 0 on a Cold reset.

Accessing RMR

When EL3 is implemented, Arm deprecates accessing this register from any PE mode other than Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

`MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}`

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b0000 | 0b010 |

```
if PSTATE.EL IN {EL1, EL3} && IsHighestEL(PSTATE.EL) then
    return RMR;
else
    UNDEFINED;
```

`MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}`

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b0000 | 0b010 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if IsHighestEL(EL1) then
        RMR = R[t];
    else
        UNDEFINED;
elseif PSTATE.EL == EL2 then
    UNDEFINED;
elseif PSTATE.EL == EL3 then
    if CP15SDISABLE == HIGH then
        UNDEFINED;
    elseif CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        RMR = R[t];
```

RVBAR, Reset Vector Base Address Register

The RVBAR characteristics are:

Purpose

If EL3 is not implemented, contains the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in AArch32 state.

Configuration

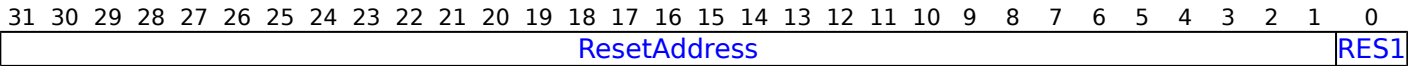
This register is present only when AArch32 is supported. Otherwise, direct accesses to RVBAR are UNDEFINED.

This register is only implemented if the highest Exception level implemented is capable of using AArch32, and is not EL3.

Attributes

RVBAR is a 32-bit register.

Field descriptions



ResetAddress, bits [31:1]

Bits [31:1] of the IMPLEMENTATION DEFINED address that execution starts from after reset when executing in 32-bit state.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bit [0]

Reserved, RES1.

Accessing RVBAR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if IsHighestEL(EL1) then
        return RVBAR;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    if IsHighestEL(EL2) then
        return RVBAR;
    else
        UNDEFINED;
elsif PSTATE.EL == EL3 then
    return MVBAR;

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCR, Secure Configuration Register

The SCR characteristics are:

Purpose

When EL3 is implemented and can use AArch32, defines the configuration of the current Security state. It specifies:

- The Security state, either Secure or Non-secure.
- What mode the PE branches to if an IRQ, FIQ, or External abort occurs.
- Whether the PSTATE.F or PSTATE.A bits can be modified when SCR.NS==1.

Configuration

AArch32 System register SCR bits [31:0] can be mapped to AArch64 System register [SCR_EL3\[31:0\]](#), but this is not architecturally mandated.

This register is present only when AArch32 is supported. Otherwise, direct accesses to SCR are UNDEFINED.

Attributes

SCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|------|-----|-----|------|-----|-----|-----|-----|----|----|----|-----|-----|----|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | TERR | RES0 | TWE | TWI | RES0 | SIF | HCE | SCD | nET | AW | FW | EA | FIQ | IRQ | NS | |

Bits [31:16]

Reserved, RES0.

TERR, bit [15]

When FEAT_RAS is implemented:

Trap Error record accesses. Generate a Monitor Trap exception on accesses to the following registers from modes other than Monitor mode:

[ERRIDR](#), [ERRSELR](#), [ERXADDR](#), [ERXADDR2](#), [ERXCTLR](#), [ERXCTLR2](#), [ERXFR](#), [ERXFR2](#), [ERXMISC0](#), [ERXMISC1](#), [ERXMISC2](#), [ERXMISC3](#), and [ERXSTATUS](#). When FEAT_RASv1p1 is implemented, [ERXMISC4](#), [ERXMISC5](#), [ERXMISC6](#), [ERXMISC7](#).

| TERR | Meaning |
|------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Accesses to the specified registers from modes other than Monitor mode generate a Monitor Trap exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [14]

Reserved, RES0.

TWE, bit [13]

Traps WFE instructions to Monitor mode.

| TWE | Meaning |
|------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any attempt to execute a WFE instruction in any mode other than Monitor mode is trapped to Monitor mode, if the instruction would otherwise have caused the PE to enter a low-power state and the attempted execution does not generate an exception that is taken to EL1 or EL2 by SCTLR.nTWE or HCR.TWE . Any exception that is taken to EL1 or to EL2 has priority over this trap. |

The attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

TWI, bit [12]

Traps WFI instructions to Monitor mode.

| TWI | Meaning |
|------------|---|
| 0b0 | This control does not cause any instructions to be trapped. |
| 0b1 | Any attempt to execute a WFI instruction in any mode other than Monitor mode is trapped to Monitor mode, if the instruction would otherwise have caused the PE to enter a low-power state and the attempted execution does not generate an exception that is taken to EL1 or EL2 by SCTLR.nTWI or HCR.TWI . Any exception that is taken to EL1 or to EL2 has priority over this trap. |

The attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Bits [11:10]

Reserved, RES0.

SIF, bit [9]

Secure instruction fetch. When the PE is in Secure state, this bit disables instruction fetch from Non-secure memory.

| SIF | Meaning |
|------------|--|
| 0b0 | Secure state instruction fetches from Non-secure memory are permitted. |
| 0b1 | Secure state instruction fetches from Non-secure memory are not permitted. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

HCE, bit [8]

Hypervisor Call instruction enable. If EL2 is implemented, enables execution of HVC instructions at Non-secure EL1 and EL2.

| HCE | Meaning |
|------------|--|
| 0b0 | HVC instructions are: <ul style="list-style-type: none"> UNDEFINED at Non-secure EL1. The Undefined Instruction exception is taken from PL1 to PL1. UNPREDICTABLE at EL2. Behavior is one of the following: <ul style="list-style-type: none"> The instruction is UNDEFINED. The instruction executes as a NOP. |
| 0b1 | HVC instructions are enabled at Non-secure EL1 and EL2. |

Note

HVC instructions are always UNDEFINED at EL0 and in Secure state.

If EL2 is not implemented, this bit is RES0 and HVC is UNDEFINED.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

SCD, bit [7]

Secure Monitor Call disable. Disables SMC instructions.

| SCD | Meaning |
|------------|--|
| 0b0 | SMC instructions are enabled. |
| 0b1 | In Non-secure state, SMC instructions are UNDEFINED. The Undefined Instruction exception is taken from the current Exception level to the current Exception level. In Secure state, behavior is one of the following: <ul style="list-style-type: none"> The instruction is UNDEFINED. The instruction executes as a NOP. |

Note

SMC instructions are always UNDEFINED at PL0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

nET, bit [6]

Not Early Termination. This bit disables early termination.

| nET | Meaning |
|------------|--|
| 0b0 | Early termination permitted. Execution time of data operations can depend on the data values. |
| 0b1 | Disable early termination. The number of cycles required for data operations is forced to be independent of the data values. |

This IMPLEMENTATION DEFINED mechanism can disable data dependent timing optimizations from multiplies and data operations. It can provide system support against information leakage that might be exploited by timing correlation types of attack.

On implementations that do not support early termination or do not support disabling early termination, this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

AW, bit [5]

When the value of SCR.EA is 1 and the value of [HCR](#).AMO is 0, this bit controls whether PSTATE.A masks an External abort taken from Non-secure state.

| AW | Meaning |
|-----------|--|
| 0b0 | External aborts taken from Non-secure state are not masked by PSTATE.A, and are taken to EL3. External aborts taken from Secure state are masked by PSTATE.A. |
| 0b1 | External aborts taken from either Security state are masked by PSTATE.A. When PSTATE.A is 0, the abort is taken to EL3. |

When SCR.EA is 0 or [HCR](#).AMO is 1, this bit has no effect.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

FW, bit [4]

When the value of SCR.FIQ is 1 and the value of [HCR](#).FMO is 0, this bit controls whether PSTATE.F masks an FIQ interrupt taken from Non-secure state.

| FW | Meaning |
|-----------|---|
| 0b0 | An FIQ taken from Non-secure state is not masked by PSTATE.F, and is taken to EL3. An FIQ taken from Secure state is masked by PSTATE.F. |
| 0b1 | An FIQ taken from either Security state is masked by PSTATE.F. When PSTATE.F is 0, the FIQ is taken to EL3. |

When SCR.FIQ is 0 or [HCR](#).FMO is 1, this bit has no effect.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

EA, bit [3]

External Abort handler. This bit controls which mode takes External aborts and SError interrupt exceptions.

| EA | Meaning |
|-----------|--|
| 0b0 | External aborts taken to Abort mode. |
| 0b1 | External aborts taken to Monitor mode. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

FIQ, bit [2]

FIQ handler. This bit controls which mode takes FIQ exceptions.

| FIQ | Meaning |
|-----|-----------------------------|
| 0b0 | FIQs taken to FIQ mode. |
| 0b1 | FIQs taken to Monitor mode. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

IRQ, bit [1]

IRQ handler. This bit controls which mode takes IRQ exceptions.

| IRQ | Meaning |
|-----|-----------------------------|
| 0b0 | IRQs taken to IRQ mode. |
| 0b1 | IRQs taken to Monitor mode. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

NS, bit [0]

Non-secure bit. Except when the PE is in Monitor mode, this bit determines the Security state of the PE:

| NS | Meaning |
|-----|----------------------------|
| 0b0 | PE is in Secure state. |
| 0b1 | PE is in Non-secure state. |

If the [HCR.TGE](#) bit is set, an attempt to change from a Secure PL1 mode to a Non-secure EL1 mode by changing the SCR.NS bit from 0 to 1 results in the SCR.NS bit remaining as 0.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Accessing SCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SCR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0001 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    SCR = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SCTLR, System Control Register

The SCTLR characteristics are:

Purpose

Provides the top level control of the system, including its memory system.

Configuration

AArch32 System register SCTLR bits [31:0] are architecturally mapped to AArch64 System register [SCTLR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to SCTLR are UNDEFINED.

Some bits in the register are read-only. These bits relate to non-configurable features of an implementation, and are provided for compatibility with previous versions of the architecture.

Attributes

SCTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|-----|-----|------|----|------|------|------|------|------|-----|------|------|------|------|----|----|------|--------|------|------|-------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 |
| DSSBS | TE | AFE | TRE | RES0 | EE | RES0 | SPAN | RES1 | RES0 | UWXN | WXN | nTWE | RES0 | nTWI | RES0 | V | I | RES1 | EnRCTX | RES0 | SEDI | TDUNK | | | |

DSSBS, bit [31]

When FEAT_SSBS is implemented:

Default PSTATE.SSBS value on Exception Entry. The defined values are:

| DSSBS | Meaning |
|-------|--|
| 0b0 | PSTATE.SSBS is set to 0 on an exception to any mode in this security state except Hyp mode |
| 0b1 | PSTATE.SSBS is set to 1 on an exception to any mode in this security state except Hyp mode |

Note

When EL3 is implemented and is using AArch32, this bit is banked between the two Security states.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

TE, bit [30]

T32 Exception Enable. This bit controls whether exceptions to an Exception level that is executing at PL1 are taken to A32 or T32 state:

| TE | Meaning |
|-----|--|
| 0b0 | Exceptions, including reset, taken to A32 state. |
| 0b1 | Exceptions, including reset, taken to T32 state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

AFE, bit [29]

Access Flag Enable. When using the Short-descriptor translation table format for the PL1&0 translation regime, this bit enables use of the AP[0] bit in the translation descriptors as the Access flag, and restricts access permissions in the translation descriptors to the simplified model.

| AFE | Meaning |
|-----|---|
| 0b0 | In the translation table descriptors, AP[0] is an access permissions bit. The full range of access permissions is supported. No Access flag is implemented. |
| 0b1 | In the translation table descriptors, AP[0] is the Access flag. Only the simplified model for access permissions is supported. |

When using the Long-descriptor translation table format, the VMSA behaves as if this bit is set to 1, regardless of the value of this bit.

The AFE bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

TRE, bit [28]

TEX remap enable. This bit enables remapping of the TEX[2:1] bits in the PL1&0 translation regime for use as two translation table bits that can be managed by the operating system. Enabling this remapping also changes the scheme used to describe the memory region attributes in the VMSA.

| TRE | Meaning |
|-----|--|
| 0b0 | TEX remap disabled. TEX[2:0] are used, with the C and B bits, to describe the memory region attributes. |
| 0b1 | TEX remap enabled. TEX[2:1] are reassigned for use as bits managed by the operating system. The TEX[0], C, and B bits are used to describe the memory region attributes, with the MMU remap registers. |

When the value of [TTBCR.EAE](#) is 1, this bit is RES1.

The TRE bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [27:26]

Reserved, RES0.

EE, bit [25]

The value of the PSTATE.E bit on branch to an exception vector or coming out of reset, and the endianness of stage 1 translation table walks in the PL1&0 translation regime.

| EE | Meaning |
|-----|---|
| 0b0 | Little-endian. PSTATE.E is cleared to 0 on taking an exception or coming out of reset. Stage 1 translation table walks in the PL1&0 translation regime are little-endian. |
| 0b1 | Big-endian. PSTATE.E is set to 1 on taking an exception or coming out of reset. Stage 1 translation table walks in the PL1&0 translation regime are big-endian. |

If an implementation does not provide Big-endian support for data accesses at Exception levels higher than EL0, this bit is RES0.

If an implementation does not provide Little-endian support for data accesses at Exception levels higher than EL0, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Bit [24]

Reserved, RES0.

SPAN, bit [23]

When FEAT_PAN is implemented:

Set Privileged Access Never, on taking an exception to EL1 from either Secure or Non-secure state, or to EL3 from Secure state when EL3 is using AArch32.

| SPAN | Meaning |
|------|---|
| 0b0 | PSTATE.PAN is set to 1 in the following situations: <ul style="list-style-type: none"> In Non-secure state, on taking an exception to EL1. In Secure state, when EL3 is using AArch64, on taking an exception to EL1. In Secure state, when EL3 is using AArch32, on taking an exception to EL3. |
| 0b1 | The value of PSTATE.PAN is left unchanged on taking an exception to EL1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bit [22]

Reserved, RES1.

Bit [21]

Reserved, RES0.

UWXN, bit [20]

Unprivileged write permission implies PL1 XN (Execute-never). This bit can force all memory regions that are writable at PL0 to be treated as XN for accesses from software executing at PL1.

| UWXN | Meaning |
|------|--|
| 0b0 | This control has no effect on memory access permissions. |
| 0b1 | Any region that is writable at PL0 forced to XN for accesses from software executing at PL1. |

The UWXN bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

WXN, bit [19]

Write permission implies XN (Execute-never). For the PL1&0 translation regime, this bit can force all memory regions that are writable to be treated as XN.

| WXN | Meaning |
|-----|---|
| 0b0 | This control has no effect on memory access permissions. |
| 0b1 | Any region that is writable in the PL1&0 translation regime is forced to XN for accesses from software executing at PL1 or PL0. |

This bit applies only when SCTLR.M bit is set.

The WXN bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

nTWE, bit [18]

Traps EL0 execution of WFE instructions to Undefined mode.

| nTWE | Meaning |
|------|---|
| 0b0 | Any attempt to execute a WFE instruction at EL0 is trapped to Undefined mode, if the instruction would otherwise have caused the PE to enter a low-power state. |
| 0b1 | This control does not cause any instructions to be trapped. |

The attempted execution of a conditional WFE instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

Bit [17]

Reserved, RES0.

nTWI, bit [16]

Traps EL0 execution of WFI instructions to Undefined mode.

| nTWI | Meaning |
|------|---|
| 0b0 | Any attempt to execute a WFI instruction at EL0 is trapped to Undefined mode, if the instruction would otherwise have caused the PE to enter a low-power state. |
| 0b1 | This control does not cause any instructions to be trapped. |

The attempted execution of a conditional WFI instruction is only trapped if the instruction passes its condition code check.

Note

Since a WFE or WFI can complete at any time, even without a Wakeup event, the traps on WFE or WFI are not guaranteed to be taken, even if the WFE or WFI is executed when there is no Wakeup event. The only guarantee is that if the instruction does not complete in finite time in the absence of a Wakeup event, the trap will be taken.

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

Bits [15:14]

Reserved, RES0.

V, bit [13]

Vectors bit. This bit selects the base address of the exception vectors for exceptions taken to a PE mode other than Monitor mode or Hyp mode:

| V | Meaning |
|-----|---|
| 0b0 | Normal exception vectors. Base address is held in VBAR . |
| 0b1 | High exception vectors (Hivecs), base address 0xFFFF0000. This base address cannot be remapped. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

I, bit [12]

Instruction access Cacheability control, for accesses at EL1 and EL0:

| I | Meaning |
|-----|---|
| 0b0 | All instruction access to Normal memory from PL1 and PL0 are Non-cacheable for all levels of instruction and unified cache. If the value of SCTLR.M is 0, instruction accesses from stage 1 of the PL1&0 translation regime are to Normal, Outer Shareable, Inner Non-cacheable, Outer Non-cacheable memory. |
| 0b1 | All instruction access to Normal memory from PL1 and PL0 can be cached at all levels of instruction and unified cache. If the value of SCTLR.M is 0, instruction accesses from stage 1 of the PL1&0 translation regime are to Normal, Outer Shareable, Inner Write-Through, Outer Write-Through memory. |

Instruction accesses to Normal memory from EL1 and EL0 are Cacheable regardless of the value of the SCTLR.I bit if either:

- EL2 is using AArch32 and the value of [HCR.DC](#) is 1.
- EL2 is using AArch64 and the value of [HCR_EL2.DC](#) is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bit [11]

Reserved, RES1.

EnRCTX, bit [10]
When FEAT_SPECPRES is implemented:

Enable EL0 access to the AArch32 CFPRCTX, DVPRCTX, and CPPRCTX instructions.

| EnRCTX | Meaning |
|--------|--|
| 0b0 | EL0 access to these instructions is disabled, and these instructions are trapped to EL1. |
| 0b1 | EL0 access to these instructions is enabled. |

Note

When EL3 is implemented and is using AArch32, this bit is banked between the two Security states.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [9]

Reserved, RES0.

SED, bit [8]

SETEND instruction disable. Disables SETEND instructions at PL0 and PL1.

| SED | Meaning |
|-----|---|
| 0b0 | SETEND instruction execution is enabled at PL0 and PL1. |
| 0b1 | SETEND instructions are UNDEFINED at PL0 and PL1. |

If the implementation does not support mixed-endian operation at any Exception level, this bit is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

ITD, bit [7]

IT Disable. Disables some uses of IT instructions at PL1 and PL0.

| ITD | Meaning |
|-----|---|
| 0b0 | All IT instruction functionality is enabled at PL1 and PL0. |
| 0b1 | Any attempt at PL1 or PL0 to execute any of the following is UNDEFINED: <ul style="list-style-type: none"> All encodings of the IT instruction with hw1[3:0]!=1000. All encodings of the subsequent instruction with the following values for hw1: <ul style="list-style-type: none"> 11xxxxxxxxxxxx: All 32-bit instructions, and the 16-bit instructions B, UDF, SVC, LDM, and STM. 1011xxxxxxxxxxxx: All instructions in 'Miscellaneous 16-bit instructions'. 10100xxxxxxxxxxx: ADD Rd, PC, #imm 01001xxxxxxxxxxx: LDR Rd, [PC, #imm] 0100x1xxx1111xxx: ADD Rdn, PC; CMP Rn, PC; MOV Rd, PC; BX PC; BLX PC. 010001xx1xxxx111: ADD PC, Rm; CMP PC, Rm; MOV PC, Rm. This pattern also covers unpredictable cases with BLX Rn. <p>These instructions are always UNDEFINED, regardless of whether they would pass or fail the condition code check that applies to them as a result of being in an IT block.</p> <p>It is IMPLEMENTATION DEFINED whether the IT instruction is treated as:</p> <ul style="list-style-type: none"> A 16-bit instruction, that can only be followed by another 16-bit instruction. The first half of a 32-bit instruction. <p>This means that, for the situations that are UNDEFINED, either the second 16-bit instruction or the 32-bit instruction is UNDEFINED.</p> <p>An implementation might vary dynamically as to whether IT is treated as a 16-bit instruction or the first half of a 32-bit instruction.</p> |

If an instruction in an active IT block that would be disabled by this field sets this field to 1 then behavior is CONSTRAINED UNPREDICTABLE. For more information see 'Changes to an ITD control by an instruction in an IT block'.

ITD is optional, but if it is implemented in the SCTLR then it must also be implemented in the [SCTLR_EL1](#), [SCTLR_EL2](#), and [HSCTLR](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

When an implementation does not implement ITD, access to this field is **RAZ/WI**.

UNK, bit [6]

Writes to this bit are IGNORED. Reads of this bit return an UNKNOWN value.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CP15BEN, bit [5]

System instruction memory barrier enable. Enables accesses to the DMB, DSB, and ISB System instructions in the (coproc==0b1111) encoding space from PL1 and PL0:

| CP15BEN | Meaning |
|---------|---|
| 0b0 | PL0 and PL1 execution of the CP15DMB , CP15DSB , and CP15ISB instructions is UNDEFINED. |
| 0b1 | PL0 and PL1 execution of the CP15DMB , CP15DSB , and CP15ISB instructions is enabled. |

CP15BEN is optional, but if it is implemented in the SCTLR then it must also be implemented in the [SCTLR_EL1](#), [SCTLR_EL2](#), and [HSCTLR](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

When an implementation does not implement CP15BEN, access to this field is **RAO/WI**.

LSMAOE, bit [4]

When FEAT_LSMAOC is implemented:

Load Multiple and Store Multiple Atomicity and Ordering Enable.

| LSMAOE | Meaning |
|--------|--|
| 0b0 | For all memory accesses at EL1 or EL0, A32 and T32 Load Multiple and Store Multiple can have an interrupt taken during the sequence memory accesses, and the memory accesses are not required to be ordered. |
| 0b1 | The ordering and interrupt behavior of A32 and T32 Load Multiple and Store Multiple at EL1 or EL0 is as defined for Armv8.0. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

Otherwise:

Reserved, RES1.

nTLSMD, bit [3]

When FEAT_LSMAOC is implemented:

No Trap Load Multiple and Store Multiple to Device-nGRE/Device-nGnRE/Device-nGnRnE memory.

| nTLSMD | Meaning |
|--------|---|
| 0b0 | All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL1 or EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are trapped and generate a stage 1 Alignment fault. |
| 0b1 | All memory accesses by A32 and T32 Load Multiple and Store Multiple at EL1 or EL0 that are marked at stage 1 as Device-nGRE/Device-nGnRE/Device-nGnRnE memory are not trapped. |

This bit is permitted to be cached in a TLB.

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

Otherwise:

Reserved, RES1.

C, bit [2]

Cacheability control, for data accesses at EL1 and EL0:

| C | Meaning |
|-----|--|
| 0b0 | All data access to Normal memory from PL1 and PL0, and all accesses to the PL1&0 stage 1 translation tables, are Non-cacheable for all levels of data and unified cache. |
| 0b1 | All data access to Normal memory from PL1 and PL0, and all accesses to the PL1&0 stage 1 translation tables, can be cached at all levels of data and unified cache. |

The PE ignores SCTLR.C for Non-secure state and data accesses to Normal memory from EL1 and EL0 are Cacheable if either:

- EL2 is using AArch32 and the value of [HCR.DC](#) is 1.
- EL2 is using AArch64 and the value of [HCR_EL2.DC](#) is 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

A, bit [1]

Alignment check enable. This is the enable bit for Alignment fault checking at PL1 and PL0:

| A | Meaning |
|-----|---|
| 0b0 | Alignment fault checking disabled when executing at PL1 or PL0. Instructions that load or store one or more registers, other than load/store exclusive and load-acquire/store-release, do not check that the address being accessed is aligned to the size of the data element(s) being accessed. |
| 0b1 | Alignment fault checking enabled when executing at PL1 or PL0. All instructions that load or store one or more registers have an alignment check that the address being accessed is aligned to the size of the data element(s) being accessed. If this check fails it causes an Alignment fault, which is taken as a Data Abort exception. |

Load/store exclusive and load-acquire/store-release instructions have an alignment check regardless of the value of the A bit.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

M, bit [0]

MMU enable for EL1 and EL0 stage 1 address translation. Possible values of this bit are:

| M | Meaning |
|-----|---|
| 0b0 | EL1 and EL0 stage 1 address translation disabled. See the SCTLR.I field for the behavior of instruction accesses to Normal memory. |
| 0b1 | EL1 and EL0 stage 1 address translation enabled. |

In the Non-secure state the PE behaves as if the value of the SCTLR.M field is 0 for all purposes other than returning the value of a direct read of the field if either:

- EL2 is using AArch32 and the value of [HCR.{DC, TGE}](#) is not {0, 0}.
- EL2 is using AArch64 and the value of [HCR_EL2.{DC, TGE}](#) is not {0, 0}.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing SCTLR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return SCTLR_NS;
    else
        return SCTLR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return SCTLR_NS;
    else
        return SCTLR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return SCTLR_S;
    else
        return SCTLR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        SCTLR_NS = R[t];
    else
        SCTLR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        SCTLR_NS = R[t];
    else
        SCTLR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            SCTLR_S = R[t];
        else
            SCTLR_NS = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SDCR, Secure Debug Control Register

The SDCR characteristics are:

Purpose

Provides EL3 configuration options for self-hosted debug, trace, and the Performance Monitors Extension.

Configuration

AArch32 System register SDCR bits [31:0] can be mapped to AArch64 System register [MDCR_EL3\[31:0\]](#), but this is not architecturally mandated.

This register is present only when AArch32 is supported. Otherwise, direct accesses to SDCR are UNDEFINED.

Attributes

SDCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-------|------|------|------|------|------|------|------|-----|------|------|-----|------|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | MTPME | TDCC | RES0 | SCCD | RES0 | EPMA | EDAD | TTRF | STE | SPME | RES0 | SPD | RES0 | | | | | | | | | | | | | | | | | | |

Bits [31:29]

Reserved, RES0.

MTPME, bit [28]

When FEAT_MTPMU is implemented:

Multi-threaded PMU Enable. Enables use of the [PMEVTYPER<n>](#).MT bits.

| MTPME | Meaning |
|-------|---|
| 0b0 | FEAT_MTPMU is disabled. The Effective value of PMEVTYPER<n> .MT is 0. |
| 0b1 | PMEVTYPER<n> .MT bits not affected by this bit. |

If FEAT_MTPMU is disabled for any other PE in the system that has the same level 1 Affinity as the PE, it is IMPLEMENTATION DEFINED whether the PE behaves as if this bit is 0.

The reset behavior of this field is:

- On a Cold reset, in a system where the PE resets into EL3, this field resets to 1.

Otherwise:

Reserved, RES0.

TDCC, bit [27]

When FEAT_FGT is implemented:

Trap DCC. Traps use of the Debug Comms Channel in modes other than Monitor mode to Monitor mode.

| TDCC | Meaning |
|-------------|--|
| 0b0 | This control does not cause any register accesses to be trapped. |
| 0b1 | Accesses to the DCC registers in modes other than Monitor mode generate a Monitor Trap exception, unless the access also generates a higher priority exception. Traps on the DCC data transfer registers are ignored when the PE is in Debug state. |

The DCC registers trapped by this control are:

- [DBGDTRRXext](#), [DBGDTRTXext](#), [DBGDSCRint](#), [DBGDCCINT](#), and, when the PE is in Non-debug state, [DBGDTRRXint](#) and [DBGDTRTXint](#).

When the PE is in Debug state, SDCR.TDCC does not trap any accesses to:

- [DBGDTRRXint](#) and [DBGDTRTXint](#).

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [26:24]

Reserved, RES0.

SCCD, bit [23]

When FEAT_PMUv3p5 is implemented:

Secure Cycle Counter Disable. Prohibits [PMCCNTR](#) from counting in Secure state.

| SCCD | Meaning |
|-------------|--|
| 0b0 | Cycle counting by PMCCNTR is not affected by this mechanism. |
| 0b1 | Cycle counting by PMCCNTR is prohibited in Secure state. |

This field does not affect the CPU_CYCLES event or any other event that counts cycles.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [22]

Reserved, RES0.

EPMAD, bit [21]

When FEAT_Debugv8p4 is implemented and FEAT_PMUv3 is implemented:

External Performance Monitors Non-secure access disable. Controls Non-secure access to Performance Monitors registers by an external debugger.

| EPMAD | Meaning |
|--------------|---|
| 0b0 | Non-secure access to the Performance Monitors registers from an external debugger is permitted. |
| 0b1 | Non-secure access to the Performance Monitors registers from an external debugger is not permitted. |

If the Performance Monitors Extension does not support external debug interface accesses, this bit is RES0.

Otherwise, if EL3 is not implemented and the Effective value of [SCR.NS](#) is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

When FEAT_PMuV3 is implemented:

External Performance Monitors access disable. Controls access to Performance Monitors registers by an external debugger.

| EPMAD | Meaning |
|--------------|--|
| 0b0 | Access to Performance Monitors registers from an external debugger is permitted. |
| 0b1 | Access to Performance Monitors registers from an external debugger is not permitted, unless overridden by the IMPLEMENTATION DEFINED authentication interface. |

If the Performance Monitors Extension does not support external debug interface accesses, this bit is RES0.

Otherwise, if EL3 is not implemented and the Effective value of [SCR.NS](#) is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

EDAD, bit [20]

When FEAT_Debugv8p4 is implemented:

External debug Non-secure access disable. Controls Non-secure access to breakpoint, watchpoint, and [OSLAR_EL1](#) registers by an external debugger.

| EDAD | Meaning |
|-------------|--|
| 0b0 | Non-secure access to debug registers from an external debugger is permitted. |
| 0b1 | Non-secure access to breakpoint registers, watchpoint registers, and OSLAR_EL1 from an external debugger is not permitted. |

If EL3 is not implemented and the Effective value of [SCR.NS](#) is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

When FEAT_Debugv8p2 is implemented:

External debug access disable. Controls access to breakpoint, watchpoint, and [OSLAR_EL1](#) registers by an external debugger.

| EDAD | Meaning |
|------|---|
| 0b0 | Access to debug registers from an external debugger is permitted. |
| 0b1 | Access to breakpoint registers, watchpoint registers, and OSLAR_EL1 from an external debugger is not permitted, unless overridden by the IMPLEMENTATION DEFINED authentication interface. |

If EL3 is not implemented and the Effective value of [SCR.NS](#) is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Otherwise:

External debug access disable. Controls access to breakpoint, watchpoint, and optionally [OSLAR_EL1](#) registers by an external debugger.

| EDAD | Meaning |
|------|--|
| 0b0 | Access to debug registers from an external debugger is permitted. |
| 0b1 | Access to breakpoint registers and watchpoint registers from an external debugger is not permitted, unless overridden by the IMPLEMENTATION DEFINED authentication interface. It is IMPLEMENTATION DEFINED whether access to the OSLAR_EL1 register from an external debugger is permitted or not permitted. |

If EL3 is not implemented and the Effective value of [SCR.NS](#) is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

TTRF, bit [19]

When FEAT_TRF is implemented:

Trap Trace Filter controls. Controls whether accesses in modes other than Monitor mode to the trace filter control registers generate a Monitor Trap exception.

| TTRF | Meaning |
|------|--|
| 0b0 | Accesses to HTRECR and TRECR are not affected by this control bit. |
| 0b1 | When not in Monitor mode, accesses to HTRECR and TRECR generate a Monitor Trap exception, unless the access generates a higher priority exception. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

STE, bit [18]

When FEAT_TRF is implemented:

Secure Trace Enable. This bit enables tracing in Secure state and controls the level of authentication required by an external debugger to enable external tracing.

| STE | Meaning |
|-----|---|
| 0b0 | Trace is prohibited in Secure state unless overridden by the IMPLEMENTATION DEFINED authentication interface. |
| 0b1 | Trace in Secure state is not affected by this bit. |

This bit also controls the level of authentication required by an external debugger to enable external tracing. See 'Register controls to enable self-hosted trace'.

If EL3 is not implemented and the Effective value of [SCR.NS](#) is 0, the PE behaves as if this bit is set to 1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

SPME, bit [17]

When FEAT_PMUv3 is implemented and FEAT_Debugv8p2 is implemented:

Secure Performance Monitors Enable. Controls event counting in Secure state.

| SPME | Meaning |
|------|---|
| 0b0 | Event counting is prohibited in Secure state. If PMCR.DP is 1, PMCCNTR is disabled in Secure state. Otherwise, PMCCNTR is not affected by this mechanism. |
| 0b1 | Event counting and PMCCNTR are not affected by this mechanism. |

This field affects the operation of all event counters in Secure state, and if [PMCR.DP](#) is 1, the operation of [PMCCNTR](#) in Secure state. When [PMCR.DP](#) is 0, [PMCCNTR](#) is not affected by this field.

If EL3 is not implemented and the Effective value of [SCR.NS](#) is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

When FEAT_PMUv3 is implemented:

Secure Performance Monitors Enable. Controls event counting in Secure state.

| SPME | Meaning |
|------|--|
| 0b0 | If ExternalSecureNoninvasiveDebugEnabled() is FALSE, event counting is prohibited in Secure state, and if PMCR.DP is 1, PMCCNTR is disabled in Secure state. |
| 0b1 | Event counting and PMCCNTR are not affected by this mechanism. |

If ExternalSecureNoninvasiveDebugEnabled() is TRUE, the event counters and [PMCCNTR](#) are not affected by this field.

Otherwise, this field affects the operation of all event counters in Secure state, and if [PMCR.DP](#) is 1, the operation of [PMCCNTR](#) in Secure state. When [PMCR.DP](#) is 0, [PMCCNTR](#) is not affected by this field.

If EL3 is not implemented and the Effective value of [SCR.NS](#) is 0, then the Effective value of this field is 1.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [16]

Reserved, RES0.

SPD, bits [15:14]

AArch32 Secure self-hosted Privileged Debug. Enables or disables debug exceptions from EL3, other than Breakpoint Instruction exceptions.

| SPD | Meaning |
|------|---|
| 0b00 | Legacy mode. Debug exceptions from EL3 are enabled by the authentication interface. |
| 0b10 | Secure privileged debug disabled. Debug exceptions from EL3 are disabled. |
| 0b11 | Secure privileged debug enabled. Debug exceptions from EL3 are enabled. |

Other values are reserved, and have the CONSTRAINED UNPREDICTABLE behavior that they must have the same behavior as 0b00. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

This field has no effect on Breakpoint Instruction exceptions. These are always enabled.

This field is ignored in Non-secure state.

If debug exceptions from EL3 are enabled, then debug exceptions from Secure EL0 are also enabled.

Otherwise, debug exceptions from Secure EL0 are enabled only if the value of [SDER.SUIDEN](#) is 1.

If EL3 is not implemented and the Effective value of [SCR.NS](#) is 0, then the Effective value of this field is 0b11.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL3, this field resets to 0.

Bits [13:0]

Reserved, RES0.

Accessing SDCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SDCR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !ELUsingAArch32(EL2) && SCR_EL3.<NS,EEL2> == '01' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif !ELUsingAArch32(EL3) && SCR_EL3.NS == '0' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        SDCR = R[t];

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SDER, Secure Debug Enable Register

The SDER characteristics are:

Purpose

Controls invasive and non-invasive debug in the Secure EL0 mode.

Configuration

AArch32 System register SDER bits [31:0] are architecturally mapped to AArch64 System register [SDER32_EL2\[31:0\]](#) when EL2 is implemented and FEAT_SEL2 is implemented.

AArch32 System register SDER bits [31:0] are architecturally mapped to AArch64 System register [SDER32_EL3\[31:0\]](#) when EL3 is implemented.

This register is present only when (EL3 is implemented and EL3 is capable of using AArch32) or (EL1 is capable of using AArch32 and Secure EL1 is implemented). Otherwise, direct accesses to SDER are UNDEFINED.

This register is ignored by the PE when one or more of the following are true:

- The PE is in Non-secure state.
- EL1 is using AArch64.

Attributes

SDER is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|--------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | SUNIDEN | | SUIDEN | | | | | | | | | | | | | |

Bits [31:2]

Reserved, RES0.

SUNIDEN, bit [1]

Secure User Non-Invasive Debug Enable.

| SUNIDEN | Meaning |
|---------|---|
| 0b0 | This bit does not affect Performance Monitors event counting at Secure EL0 |
| 0b1 | If EL3 or EL1 is using AArch32, Performance Monitors event counting is allowed in Secure EL0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

SUIDEN, bit [0]

Secure User Invasive Debug Enable.

| SUIDEN | Meaning |
|--------|---|
| 0b0 | This bit does not affect the generation of debug exceptions at Secure EL0. |
| 0b1 | If EL3 or EL1 is using AArch32, debug exceptions from Secure EL0 are enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing SDER

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        return SDER;
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    return SDER;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif !IsSecure() then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.<TDE,TDA> != '00' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TDA == '1' then
        AArch64.AArch32SystemAccessTrap(EL3, 0x03);
    else
        SDER = R[t];
elsif PSTATE.EL == EL2 then
    UNDEFINED;
elsif PSTATE.EL == EL3 then
    if CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        SDER = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR, Saved Program Status Register

The SPSR characteristics are:

Purpose

Holds the saved process state for the current mode.

Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to SPSR are UNDEFINED.

Attributes

SPSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|---------|----|------|-----|-----|----|----|----|----|----|----|----|----|---------|----|----|----|---|---|---|---|---|---|---|--------|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| N | Z | C | V | Q | IT[1:0] | J | SSBS | PAN | DIT | IL | | GE | | | | | | IT[7:2] | | | | E | A | I | F | T | | | M[4:0] | | |

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to the current mode, and copied to PSTATE.N on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to the current mode, and copied to PSTATE.Z on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to the current mode, and copied to PSTATE.C on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to the current mode, and copied to PSTATE.V on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to the current mode, and copied to PSTATE.Q on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to the current mode, and copied to PSTATE.IT on executing an exception return operation in the current mode.

SPSR.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR[26:25].
- IT[7:2] is SPSR[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to the current mode, and copied to PSTATE.SSBS on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to the current mode, and copied to PSTATE.PAN on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to the current mode, and copied to PSTATE.DIT on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to the current mode, and copied to PSTATE.IL on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to the current mode, and copied to PSTATE.GE on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to the current mode, and copied to PSTATE.E on executing an exception return operation in the current mode.

If the implementation does not support big-endian operation, SPSR.E is RES0. If the implementation does not support little-endian operation, SPSR.E is RES1. On executing an exception return operation in the current mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to the current mode, and copied to PSTATE.A on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to the current mode, and copied to PSTATE.I on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to the current mode, and copied to PSTATE.F on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to the current mode, and copied to PSTATE.T on executing an exception return operation in the current mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to the current mode, and copied to PSTATE.M[4:0] on executing an exception return operation in the current mode.

| M[4:0] | Meaning |
|---------------|----------------|
| 0b10000 | User. |
| 0b10001 | FIQ. |
| 0b10010 | IRQ. |
| 0b10011 | Supervisor. |
| 0b10110 | Monitor. |
| 0b10111 | Abort. |
| 0b11010 | Hyp. |
| 0b11011 | Undefined. |
| 0b11111 | System. |

Other values are reserved. If SPSR.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in the current mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR

SPSR can be read using the MRS instruction and written using the MSR (register) or MSR (immediate) instructions.

SPSR_abt, Saved Program Status Register (Abort mode)

The SPSR_abt characteristics are:

Purpose

Holds the saved process state when an exception is taken to Abort mode.

Configuration

AArch32 System register SPSR_abt bits [31:0] are architecturally mapped to AArch64 System register [SPSR_abt\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to SPSR_abt are UNDEFINED.

Attributes

SPSR_abt is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|---------|----|------|-----|-----|----|----|----|----|---------|----|----|----|----|----|----|----|--------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| N | Z | C | V | Q | IT[1:0] | J | SSBS | PAN | DIT | IL | | GE | | IT[7:2] | | E | A | I | F | T | | M[4:0] | | | | | | | | | |

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Abort mode, and copied to PSTATE.N on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Abort mode, and copied to PSTATE.Z on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Abort mode, and copied to PSTATE.C on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Abort mode, and copied to PSTATE.V on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Abort mode, and copied to PSTATE.Q on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Abort mode, and copied to PSTATE.IT on executing an exception return operation in Abort mode.

SPSR_abt.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_abt[26:25].
- IT[7:2] is SPSR_abt[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Abort mode, and copied to PSTATE.SSBS on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Abort mode, and copied to PSTATE.PAN on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Abort mode, and copied to PSTATE.DIT on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Abort mode, and copied to PSTATE.IL on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Abort mode, and copied to PSTATE.GE on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Abort mode, and copied to PSTATE.E on executing an exception return operation in Abort mode.

If the implementation does not support big-endian operation, SPSR_abt.E is RES0. If the implementation does not support little-endian operation, SPSR_abt.E is RES1. On executing an exception return operation in Abort mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_abt.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_abt.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Abort mode, and copied to PSTATE.A on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Abort mode, and copied to PSTATE.I on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Abort mode, and copied to PSTATE.F on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Abort mode, and copied to PSTATE.T on executing an exception return operation in Abort mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Abort mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Abort mode.

| M[4:0] | Meaning |
|---------------|----------------|
| 0b10000 | User. |
| 0b10001 | FIQ. |
| 0b10010 | IRQ. |
| 0b10011 | Supervisor. |
| 0b10111 | Abort. |
| 0b11011 | Undefined. |
| 0b11111 | System. |

Other values are reserved. If SPSR_abt.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Abort mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_abt

SPSR_abt is accessible in all modes other than User mode and Abort mode.

Accesses to this register use the following encodings in the System register encoding space:

MRS{<c>}{<q>} <Rd>, SPSR_abt

| R | M | M1 |
|----------|----------|-----------|
| 0b1 | 0b1 | 0b0100 |

MSR{<c>}{<q>} SPSR_abt, <Rn>

| R | M | M1 |
|----------|----------|-----------|
| 0b1 | 0b1 | 0b0100 |

SPSR_fiq, Saved Program Status Register (FIQ mode)

The SPSR_fiq characteristics are:

Purpose

Holds the saved process state when an exception is taken to FIQ mode.

Configuration

AArch32 System register SPSR_fiq bits [31:0] are architecturally mapped to AArch64 System register [SPSR_fiq\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to SPSR_fiq are UNDEFINED.

Attributes

SPSR_fiq is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|---------|----|------|-----|-----|----|----|----|----|----|----|----|---------|----|----|----|----|---|---|---|---|---|--------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| N | Z | C | V | Q | IT[1:0] | J | SSBS | PAN | DIT | IL | | GE | | | | | IT[7:2] | | | | E | A | I | F | T | | M[4:0] | | | | |

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to FIQ mode, and copied to PSTATE.N on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to FIQ mode, and copied to PSTATE.Z on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to FIQ mode, and copied to PSTATE.C on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to FIQ mode, and copied to PSTATE.V on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to FIQ mode, and copied to PSTATE.Q on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to FIQ mode, and copied to PSTATE.IT on executing an exception return operation in FIQ mode.

SPSR_fiq.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_fiq[26:25].
- IT[7:2] is SPSR_fiq[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to FIQ mode, and copied to PSTATE.SSBS on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to FIQ mode, and copied to PSTATE.PAN on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to FIQ mode, and copied to PSTATE.DIT on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to FIQ mode, and copied to PSTATE.IL on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to FIQ mode, and copied to PSTATE.GE on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to FIQ mode, and copied to PSTATE.E on executing an exception return operation in FIQ mode.

If the implementation does not support big-endian operation, SPSR_fiq.E is RES0. If the implementation does not support little-endian operation, SPSR_fiq.E is RES1. On executing an exception return operation in FIQ mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_fiq.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_fiq.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to FIQ mode, and copied to PSTATE.A on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to FIQ mode, and copied to PSTATE.I on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to FIQ mode, and copied to PSTATE.F on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to FIQ mode, and copied to PSTATE.T on executing an exception return operation in FIQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to FIQ mode, and copied to PSTATE.M[4:0] on executing an exception return operation in FIQ mode.

| M[4:0] | Meaning |
|---------------|----------------|
| 0b10000 | User. |
| 0b10001 | FIQ. |
| 0b10010 | IRQ. |
| 0b10011 | Supervisor. |
| 0b10111 | Abort. |
| 0b11011 | Undefined. |
| 0b11111 | System. |

Other values are reserved. If SPSR_fiq.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in FIQ mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_fiq

SPSR_fiq is accessible in all modes other than User mode and FIQ mode.

Accesses to this register use the following encodings in the System register encoding space:

MRS{<c>}{<q>} <Rd>, SPSR_fiq

| R | M | M1 |
|----------|----------|-----------|
| 0b1 | 0b0 | 0b1110 |

MSR{<c>}{<q>} SPSR_fiq, <Rn>

| R | M | M1 |
|----------|----------|-----------|
| 0b1 | 0b0 | 0b1110 |

SPSR_hyp, Saved Program Status Register (Hyp mode)

The SPSR_hyp characteristics are:

Purpose

Holds the saved process state when an exception is taken to Hyp mode.

Configuration

AArch32 System register SPSR_hyp bits [31:0] are architecturally mapped to AArch64 System register [SPSR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to SPSR_hyp are UNDEFINED.

Attributes

SPSR_hyp is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|---------|----|------|-----|-----|----|----|----|----|----|----|----|----|---------|----|----|----|---|---|---|---|---|---|---|--------|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| N | Z | C | V | Q | IT[1:0] | J | SSBS | PAN | DIT | IL | | GE | | | | | | IT[7:2] | | | | E | A | I | F | T | | | M[4:0] | | |

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Hyp mode, and copied to PSTATE.N on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Hyp mode, and copied to PSTATE.Z on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Hyp mode, and copied to PSTATE.C on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Hyp mode, and copied to PSTATE.V on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Hyp mode, and copied to PSTATE.Q on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Hyp mode, and copied to PSTATE.IT on executing an exception return operation in Hyp mode.

SPSR_hyp.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_hyp[26:25].
- IT[7:2] is SPSR_hyp[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Hyp mode, and copied to PSTATE.SSBS on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Hyp mode, and copied to PSTATE.PAN on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Hyp mode, and copied to PSTATE.DIT on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Hyp mode, and copied to PSTATE.IL on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Hyp mode, and copied to PSTATE.GE on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Hyp mode, and copied to PSTATE.E on executing an exception return operation in Hyp mode.

If the implementation does not support big-endian operation, SPSR_hyp.E is RES0. If the implementation does not support little-endian operation, SPSR_hyp.E is RES1. On executing an exception return operation in Hyp mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_hyp.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_hyp.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Hyp mode, and copied to PSTATE.A on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Hyp mode, and copied to PSTATE.I on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Hyp mode, and copied to PSTATE.F on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Hyp mode, and copied to PSTATE.T on executing an exception return operation in Hyp mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Hyp mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Hyp mode.

| M[4:0] | Meaning |
|---------------|----------------|
| 0b10000 | User. |
| 0b10001 | FIQ. |
| 0b10010 | IRQ. |
| 0b10011 | Supervisor. |
| 0b10111 | Abort. |
| 0b11010 | Hyp. |
| 0b11011 | Undefined. |
| 0b11111 | System. |

Other values are reserved. If SPSR_hyp.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Hyp mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_hyp

SPSR_hyp is accessible only in Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

MRS{<c>}{<q>} <Rd>, SPSR_hyp

| R | M | M1 |
|----------|----------|-----------|
| 0b1 | 0b1 | 0b1110 |

SPSR_hyp, Saved Program Status Register (Hyp mode)

MSR{<c>}{<q>} SPSR_hyp, <Rn>

| R | M | M1 |
|----------|----------|-----------|
| 0b1 | 0b1 | 0b1110 |

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_irq, Saved Program Status Register (IRQ mode)

The SPSR_irq characteristics are:

Purpose

Holds the saved process state when an exception is taken to IRQ mode.

Configuration

AArch32 System register SPSR_irq bits [31:0] are architecturally mapped to AArch64 System register [SPSR_irq\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to SPSR_irq are UNDEFINED.

Attributes

SPSR_irq is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|---------|----|------|-----|-----|----|----|----|----|----|----|----|----|---------|----|----|----|---|---|---|---|---|---|--------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| N | Z | C | V | Q | IT[1:0] | J | SSBS | PAN | DIT | IL | | GE | | | | | | IT[7:2] | | | | E | A | I | F | T | | M[4:0] | | | |

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to IRQ mode, and copied to PSTATE.N on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to IRQ mode, and copied to PSTATE.Z on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to IRQ mode, and copied to PSTATE.C on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to IRQ mode, and copied to PSTATE.V on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to IRQ mode, and copied to PSTATE.Q on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to IRQ mode, and copied to PSTATE.IT on executing an exception return operation in IRQ mode.

SPSR_irq.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_irq[26:25].
- IT[7:2] is SPSR_irq[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]**When FEAT_SSBS is implemented:**

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to IRQ mode, and copied to PSTATE.SSBS on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]**When FEAT_PAN is implemented:**

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to IRQ mode, and copied to PSTATE.PAN on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to IRQ mode, and copied to PSTATE.DIT on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to IRQ mode, and copied to PSTATE.IL on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to IRQ mode, and copied to PSTATE.GE on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to IRQ mode, and copied to PSTATE.E on executing an exception return operation in IRQ mode.

If the implementation does not support big-endian operation, SPSR_irq.E is RES0. If the implementation does not support little-endian operation, SPSR_irq.E is RES1. On executing an exception return operation in IRQ mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_irq.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_irq.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to IRQ mode, and copied to PSTATE.A on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to IRQ mode, and copied to PSTATE.I on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to IRQ mode, and copied to PSTATE.F on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to IRQ mode, and copied to PSTATE.T on executing an exception return operation in IRQ mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to IRQ mode, and copied to PSTATE.M[4:0] on executing an exception return operation in IRQ mode.

| M[4:0] | Meaning |
|---------------|----------------|
| 0b10000 | User. |
| 0b10001 | FIQ. |
| 0b10010 | IRQ. |
| 0b10011 | Supervisor. |
| 0b10111 | Abort. |
| 0b11011 | Undefined. |
| 0b11111 | System. |

Other values are reserved. If SPSR_irq.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in IRQ mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_irq

SPSR_irq is accessible in all modes other than User mode and IRQ mode.

Accesses to this register use the following encodings in the System register encoding space:

MRS{<c>}{<q>} <Rd>, SPSR_irq

| R | M | M1 |
|----------|----------|-----------|
| 0b1 | 0b1 | 0b0000 |

MSR{<c>}{<q>} SPSR_irq, <Rn>

| R | M | M1 |
|----------|----------|-----------|
| 0b1 | 0b1 | 0b0000 |

SPSR_mon, Saved Program Status Register (Monitor mode)

The SPSR_mon characteristics are:

Purpose

Holds the saved process state when an exception is taken to Monitor mode.

Configuration

AArch32 System register SPSR_mon bits [31:0] can be mapped to AArch64 System register [SPSR_EL3\[31:0\]](#), but this is not architecturally mandated.

This register is present only when AArch32 is supported. Otherwise, direct accesses to SPSR_mon are UNDEFINED.

Attributes

SPSR_mon is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|---------|----|------|-----|-----|----|----|----|----|----|----|----|----|---------|----|----|----|---|---|---|---|---|---|---|--------|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| N | Z | C | V | Q | IT[1:0] | J | SSBS | PAN | DIT | IL | | GE | | | | | | IT[7:2] | | | | E | A | I | F | T | | | M[4:0] | | |

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Monitor mode, and copied to PSTATE.N on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Monitor mode, and copied to PSTATE.Z on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Monitor mode, and copied to PSTATE.C on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Monitor mode, and copied to PSTATE.V on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Monitor mode, and copied to PSTATE.Q on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Monitor mode, and copied to PSTATE.IT on executing an exception return operation in Monitor mode.

SPSR_mon.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_mon[26:25].
- IT[7:2] is SPSR_mon[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Monitor mode, and copied to PSTATE.SSBS on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Monitor mode, and copied to PSTATE.PAN on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Monitor mode, and copied to PSTATE.DIT on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Monitor mode, and copied to PSTATE.IL on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Monitor mode, and copied to PSTATE.GE on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Monitor mode, and copied to PSTATE.E on executing an exception return operation in Monitor mode.

If the implementation does not support big-endian operation, SPSR_mon.E is RES0. If the implementation does not support little-endian operation, SPSR_mon.E is RES1. On executing an exception return operation in Monitor mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_mon.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_mon.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Monitor mode, and copied to PSTATE.A on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Monitor mode, and copied to PSTATE.I on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Monitor mode, and copied to PSTATE.F on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Monitor mode, and copied to PSTATE.T on executing an exception return operation in Monitor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Monitor mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Monitor mode.

| M[4:0] | Meaning |
|---------------|----------------|
| 0b10000 | User. |
| 0b10001 | FIQ. |
| 0b10010 | IRQ. |
| 0b10011 | Supervisor. |
| 0b10110 | Monitor. |
| 0b10111 | Abort. |
| 0b11010 | Hyp. |
| 0b11011 | Undefined. |
| 0b11111 | System. |

Other values are reserved. If SPSR_mon.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Monitor mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_mon

SPSR_mon is only accessible in EL3 modes other than Monitor mode.

Accesses to this register use the following encodings in the System register encoding space:

MRS{<c>}{<q>} <Rd>, SPSR_mon

| R | M | M1 |
|----------|----------|-----------|
| 0b1 | 0b1 | 0b1100 |

SPSR_mon, Saved Program Status Register (Monitor mode)

MSR{<c>}{<q>} SPSR_mon, <Rn>

| R | M | M1 |
|----------|----------|-----------|
| 0b1 | 0b1 | 0b1100 |

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_svc, Saved Program Status Register (Supervisor mode)

The SPSR_svc characteristics are:

Purpose

Holds the saved process state when an exception is taken to Supervisor mode.

Configuration

AArch32 System register SPSR_svc bits [31:0] are architecturally mapped to AArch64 System register [SPSR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to SPSR_svc are UNDEFINED.

Attributes

SPSR_svc is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|---------|----|------|-----|-----|----|----|----|----|----|----|----|----|---------|----|----|----|---|---|---|---|---|---|---|--------|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| N | Z | C | V | Q | IT[1:0] | J | SSBS | PAN | DIT | IL | | GE | | | | | | IT[7:2] | | | | E | A | I | F | T | | | M[4:0] | | |

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Supervisor mode, and copied to PSTATE.N on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Supervisor mode, and copied to PSTATE.Z on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Supervisor mode, and copied to PSTATE.C on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Supervisor mode, and copied to PSTATE.V on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Supervisor mode, and copied to PSTATE.Q on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Supervisor mode, and copied to PSTATE.IT on executing an exception return operation in Supervisor mode.

SPSR_svc.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_svc[26:25].
- IT[7:2] is SPSR_svc[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Supervisor mode, and copied to PSTATE.SSBS on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Supervisor mode, and copied to PSTATE.PAN on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Supervisor mode, and copied to PSTATE.DIT on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Supervisor mode, and copied to PSTATE.IL on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Supervisor mode, and copied to PSTATE.GE on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Supervisor mode, and copied to PSTATE.E on executing an exception return operation in Supervisor mode.

If the implementation does not support big-endian operation, SPSR_svc.E is RES0. If the implementation does not support little-endian operation, SPSR_svc.E is RES1. On executing an exception return operation in Supervisor mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_svc.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_svc.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Supervisor mode, and copied to PSTATE.A on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Supervisor mode, and copied to PSTATE.I on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Supervisor mode, and copied to PSTATE.F on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Supervisor mode, and copied to PSTATE.T on executing an exception return operation in Supervisor mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Supervisor mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Supervisor mode.

| M[4:0] | Meaning |
|---------|-------------|
| 0b10000 | User. |
| 0b10001 | FIQ. |
| 0b10010 | IRQ. |
| 0b10011 | Supervisor. |
| 0b10111 | Abort. |
| 0b11011 | Undefined. |
| 0b11111 | System. |

Other values are reserved. If SPSR_svc.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Supervisor mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_svc

SPSR_svc is accessible in all modes other than User mode and Supervisor mode.

Accesses to this register use the following encodings in the System register encoding space:

MRS{<c>}{<q>} <Rd>, SPSR_svc

| R | M | M1 |
|-----|-----|--------|
| 0b1 | 0b1 | 0b0010 |

SPSR_svc, Saved Program Status Register (Supervisor mode)

MSR{<c>}{<q>} SPSR_svc, <Rn>

| R | M | M1 |
|----------|----------|-----------|
| 0b1 | 0b1 | 0b0010 |

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

SPSR_und, Saved Program Status Register (Undefined mode)

The SPSR_und characteristics are:

Purpose

Holds the saved process state when an exception is taken to Undefined mode.

Configuration

AArch32 System register SPSR_und bits [31:0] are architecturally mapped to AArch64 System register [SPSR_und\[31:0\]](#).
This register is present only when AArch32 is supported. Otherwise, direct accesses to SPSR_und are UNDEFINED.

Attributes

SPSR_und is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|---------|----|------|-----|-----|----|----|----|----|----|---------|----|----|----|----|----|----|---|---|---|--------|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| N | Z | C | V | Q | IT[1:0] | J | SSBS | PAN | DIT | IL | | GE | | | IT[7:2] | | E | A | I | F | T | | | | M[4:0] | | | | | | |

N, bit [31]

Negative Condition flag. Set to the value of PSTATE.N on taking an exception to Undefined mode, and copied to PSTATE.N on executing an exception return operation in Undefined mode.
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Z, bit [30]

Zero Condition flag. Set to the value of PSTATE.Z on taking an exception to Undefined mode, and copied to PSTATE.Z on executing an exception return operation in Undefined mode.
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

C, bit [29]

Carry Condition flag. Set to the value of PSTATE.C on taking an exception to Undefined mode, and copied to PSTATE.C on executing an exception return operation in Undefined mode.
The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

V, bit [28]

Overflow Condition flag. Set to the value of PSTATE.V on taking an exception to Undefined mode, and copied to PSTATE.V on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Q, bit [27]

Overflow or saturation flag. Set to the value of PSTATE.Q on taking an exception to Undefined mode, and copied to PSTATE.Q on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IT, bits [15:10, 26:25]

If-Then. Set to the value of PSTATE.IT on taking an exception to Undefined mode, and copied to PSTATE.IT on executing an exception return operation in Undefined mode.

SPSR_und.IT must contain a value that is valid for the instruction being returned to.

The IT field is split as follows:

- IT[1:0] is SPSR_und[26:25].
- IT[7:2] is SPSR_und[15:10].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

J, bit [24]

RES0.

In previous versions of the architecture, the {J, T} bits determined the AArch32 Instruction set state.

Armv8 does not support either Jazelle state or T32EE state, and the T bit determines the Instruction set state.

SSBS, bit [23]

When FEAT_SSBS is implemented:

Speculative Store Bypass. Set to the value of PSTATE.SSBS on taking an exception to Undefined mode, and copied to PSTATE.SSBS on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PAN, bit [22]

When FEAT_PAN is implemented:

Privileged Access Never. Set to the value of PSTATE.PAN on taking an exception to Undefined mode, and copied to PSTATE.PAN on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DIT, bit [21]**When FEAT_DIT is implemented:**

Data Independent Timing. Set to the value of PSTATE.DIT on taking an exception to Undefined mode, and copied to PSTATE.DIT on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IL, bit [20]

Illegal Execution state. Set to the value of PSTATE.IL on taking an exception to Undefined mode, and copied to PSTATE.IL on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

GE, bits [19:16]

Greater than or Equal flags. Set to the value of PSTATE.GE on taking an exception to Undefined mode, and copied to PSTATE.GE on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

E, bit [9]

Endianness. Set to the value of PSTATE.E on taking an exception to Undefined mode, and copied to PSTATE.E on executing an exception return operation in Undefined mode.

If the implementation does not support big-endian operation, SPSR_und.E is RES0. If the implementation does not support little-endian operation, SPSR_und.E is RES1. On executing an exception return operation in Undefined mode, if the implementation does not support big-endian operation at the Exception level being returned to, SPSR_und.E is RES0, and if the implementation does not support little-endian operation at the Exception level being returned to, SPSR_und.E is RES1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

A, bit [8]

SError interrupt mask. Set to the value of PSTATE.A on taking an exception to Undefined mode, and copied to PSTATE.A on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

I, bit [7]

IRQ interrupt mask. Set to the value of PSTATE.I on taking an exception to Undefined mode, and copied to PSTATE.I on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

F, bit [6]

FIQ interrupt mask. Set to the value of PSTATE.F on taking an exception to Undefined mode, and copied to PSTATE.F on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T, bit [5]

T32 Instruction set state. Set to the value of PSTATE.T on taking an exception to Undefined mode, and copied to PSTATE.T on executing an exception return operation in Undefined mode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

M[4:0], bits [4:0]

Mode. Set to the value of PSTATE.M[4:0] on taking an exception to Undefined mode, and copied to PSTATE.M[4:0] on executing an exception return operation in Undefined mode.

| M[4:0] | Meaning |
|---------|-------------|
| 0b10000 | User. |
| 0b10001 | FIQ. |
| 0b10010 | IRQ. |
| 0b10011 | Supervisor. |
| 0b10111 | Abort. |
| 0b11011 | Undefined. |
| 0b11111 | System. |

Other values are reserved. If SPSR_und.M[4:0] has a Reserved value, or a value for an unimplemented Exception level, executing an exception return operation in Undefined mode is an illegal return event, as described in 'Illegal return events from AArch32 state'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing SPSR_und

SPSR_und is accessible in all modes other than User mode and Undefined mode.

Accesses to this register use the following encodings in the System register encoding space:

MRS{<c>}{<q>} <Rd>, SPSR_und

| R | M | M1 |
|-----|-----|--------|
| 0b1 | 0b1 | 0b0110 |

MSR{<c>}{<q>} SPSR_und, <Rn>

| R | M | M1 |
|-----|-----|--------|
| 0b1 | 0b1 | 0b0110 |

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TCMTR, TCM Type Register

The TCMTR characteristics are:

Purpose

Provides information about the implementation of the TCM.

Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to TCMTR are UNDEFINED.

If EL1 or above can use AArch32 then this register must be implemented.

Attributes

TCMTR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing TCMTR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return TCMTR;
elsif PSTATE.EL == EL2 then
    return TCMTR;
elsif PSTATE.EL == EL3 then
    return TCMTR;

```


TLBIALL, TLB Invalidate All

The TLBIALL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at EL1, all entries that:
 - Would be required for the EL1&0 translation regime.
 - Match the current VMID, if EL2 is implemented and enabled in the current Security state.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at EL2, and if EL2 is enabled in the current Security state, the stage 1 or stage 2 translation table entries that would be required for the PL1&0 translation regime and matches the current VMID.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIALL are UNDEFINED.

Attributes

TLBIALL is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the TLBIALL instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0111 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
then
            AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_ExcludeXS);
        else
            AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_AllAttr);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
            AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_AllAttr);
        else
            if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
&& IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
                AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBI_ExcludeXS);
            else
                AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBI_AllAttr);
    elsif PSTATE.EL == EL2 then
        AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBI_AllAttr);
    elsif PSTATE.EL == EL3 then
        AArch32.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL30, Shareability_NSH, TLBI_ExcludeXS);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIALLH, TLB Invalidate All, Hyp mode

The TLBIALLH characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIALLH are UNDEFINED.

Attributes

TLBIALLH is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the TLBIALLH instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1000 | 0b0111 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_NSH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_ALL(SS_NonSecure, Regime_EL2, Shareability_NSH, TLBI_AllAttr);

```


TLBIALLHIS, TLB Invalidate All, Hyp mode, Inner Shareable

The TLBIALLHIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIALLHIS are UNDEFINED.

Attributes

TLBIALLHIS is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the TLBIALLHIS instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1000 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_ALL(SecurityStateAtEL(EL2), Regime_EL2, Shareability_ISH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_ALL(SS_NonSecure, Regime_EL2, Shareability_ISH, TLBI_AllAttr);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIALLIS, TLB Invalidate All, Inner Shareable

The TLBIALLIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk. The entries that are invalidated are as follows:

- If executed at EL1, all entries that:
 - Would be required for the EL1&0 translation regime.
 - Match the current VMID, if EL2 is implemented and enabled in the current Security state.
- If executed in Secure state when EL3 is using AArch32, all entries that would be required for the Secure PL1&0 translation regime.
- If executed at EL2, and if EL2 is enabled in the current Security state, the stage 1 or stage 2 translation table entries that would be required for the PL1&0 translation regime and matches the current VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIALLIS are UNDEFINED.

Attributes

TLBIALLIS is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the TLBIALLIS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0011 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
        && IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
            AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBI_ExcludeXS);
        else
            AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBI_AllAttr);
    endif PSTATE.EL == EL2 then
        AArch32.TLBI_VMALL(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBI_AllAttr);
    elsif PSTATE.EL == EL3 then
        AArch32.TLBI_ALL(SecurityStateAtEL(EL3), Regime_EL30, Shareability_ISH, TLBI_ExcludeXS);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIALLSNH, TLB Invalidate All, Non-Secure Non-Hyp

The TLBIALLSNH characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for stage 1 or stage 2 of the Non-secure PL1&0 translation regime, regardless of the associated VMID.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIALLSNH are UNDEFINED.

Attributes

TLBIALLSNH is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the TLBIALLSNH instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1000 | 0b0111 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_NSH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_ALL(SS_NonSecure, Regime_EL10, Shareability_NSH, TLBI_AllAttr);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIALLSNHNHIS, TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable

The TLBIALLSNHNHIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for stage 1 or stage 2 of the Non-secure PL1&0 translation regime, regardless of the associated VMID.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIALLSNHNHIS are UNDEFINED.

Attributes

TLBIALLSNHNHIS is a 32-bit System instruction.

Field descriptions

This instruction has no applicable fields.

The value in the register specified by <Rt> is ignored.

Executing the TLBIALLSNHNHIS instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1000 | 0b0011 | 0b100 |

```
if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_ALL(SecurityStateAtEL(EL1), Regime_EL10, Shareability_ISH, TLBI_AllAttr);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_ALL(SS_NonSecure, Regime_EL10, Shareability_ISH, TLBI_AllAttr);
```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIASID, TLB Invalidate by ASID match

The TLBIASID characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
 - Is from a level of lookup above the final level.
 - Is a non-global entry from the final level of lookup.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIASID are UNDEFINED.

Attributes

TLBIASID is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | ASID | | | | | | | |

Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.

Executing the TLBIASID instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0111 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
then
            AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_AllAttr, R[t]);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
            AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBI_AllAttr, R[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
&& IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
                AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBI_ExcludeXS, R[t]);
            else
                AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL2 then
        AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH, TLBI_AllAttr,
R[t]);
    elsif PSTATE.EL == EL3 then
        AArch32.TLBI_ASID(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_NSH,
TLBI_AllAttr, R[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIASIDIS, TLB Invalidate by ASID match, Inner Shareable

The TLBIASIDIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used for the specified ASID, and either:
 - Is from a level of lookup above the final level.
 - Is a non-global entry from the final level of lookup.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIASIDIS are UNDEFINED.

Attributes

TLBIASIDIS is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | ASID | | | | | | | |

Bits [31:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries for non-global pages that match the ASID values will be affected by this System instruction.

Executing the TLBIASIDIS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| | | | | |
|--------|------|-----|-----|------|
| coproc | opc1 | CRn | CRm | opc2 |
|--------|------|-----|-----|------|

| | | | | |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0011 | 0b010 |
|--------|-------|--------|--------|-------|

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
        && IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
            AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBI_AllAttr, R[t]);
        elsif PSTATE.EL == EL2 then
            AArch32.TLBI_ASID(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH, TLBI_AllAttr,
            R[t]);
        elsif PSTATE.EL == EL3 then
            AArch32.TLBI_ASID(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_ISH,
            TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIIPAS2, TLB Invalidate by Intermediate Physical Address, Stage 2

The TLBIIPAS2 characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- [SCR.NS](#) is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIIPAS2 are UNDEFINED.

Note

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIIPAS2 is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | IPA[39:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:28]

Reserved, RES0.

IPA[39:12], bits [27:0]

Bits[39:12] of the intermediate physical address to match.

Executing the TLBIIPAS2 instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1000 | 0b0100 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBIlevel_Any, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    elsif SCR.NS == '0' then
        //no operation
    else
        AArch32.TLBI_IPAS2(SS_NonSecure, Regime_EL10, VMID_NONE, Shareability_NSH, TLBIlevel_Any,
        TLBI_AllAttr, R[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIIPAS2IS, TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable

The TLBIIPAS2IS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from any level of the translation table walk.
- [SCR.NS](#) is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIIPAS2IS are UNDEFINED.

Note

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIIPAS2IS is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | IPA[39:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:28]

Reserved, RES0.

IPA[39:12], bits [27:0]

Bits[39:12] of the intermediate physical address to match.

Executing the TLBIIPAS2IS instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1000 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Any, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    elsif SCR.NS == '0' then
        //no operation
    else
        AArch32.TLBI_IPAS2(SS_NonSecure, Regime_EL10, VMID_NONE, Shareability_ISH, TLBIlevel_Any,
        TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIIPAS2L, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level

The TLBIIPAS2L characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- [SCR.NS](#) is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIIPAS2L are UNDEFINED.

Note

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIIPAS2L is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | IPA[39:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:28]

Reserved, RES0.

IPA[39:12], bits [27:0]

Bits[39:12] of the intermediate physical address to match.

Executing the TLBIIPAS2L instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1000 | 0b0100 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
    TLBIlevel_Last, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    elsif SCR.NS == '0' then
        //no operation
    else
        AArch32.TLBI_IPAS2(SS_NonSecure, Regime_EL10, VMID_NONE, Shareability_NSH, TLBIlevel_Last,
        TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIIPAS2LIS, TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable

The TLBIIPAS2LIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 2 only translation table entry, from the final level of the translation table walk.
- [SCR.NS](#) is 1.
- The entry would be used for the specified IPA.
- The entry would be used with the current VMID.
- The entry would be required for the PL1&0 translation regime.

The invalidation is not required to apply to caching structures that combine stage 1 and stage 2 translation table entries.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIIPAS2LIS are UNDEFINED.

Note

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIIPAS2LIS is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | IPA[39:12] | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:28]

Reserved, RES0.

IPA[39:12], bits [27:0]

Bits[39:12] of the intermediate physical address to match.

Executing the TLBIIPAS2LIS instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.

- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1000 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_IPAS2(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
    TLBIlevel_Last, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    elsif SCR.NS == '0' then
        //no operation
    else
        AArch32.TLBI_IPAS2(SS_NonSecure, Regime_EL10, VMID_NONE, Shareability_ISH, TLBIlevel_Last,
        TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVA, TLB Invalidate by VA

The TLBIMVA characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
 - The entry is from a level of lookup above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIMVA are UNDEFINED.

Attributes

TLBIMVA is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA | | | | | | | | | | | | RES0 | | | | ASID | | | | | | | | | | | | | | | |

VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

Executing the TLBIMVA instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0111 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
then
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
&& IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
                AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Any, TLBI_ExcludeXS, R[t]);
            else
                AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL2 then
        AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL3 then
        AArch32.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAA, TLB Invalidate by VA, All ASID

The TLBIMVAA characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIMVAA are UNDEFINED.

Attributes

TLBIMVAA is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA | | | | | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | |

VA, bits [31:12]

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVAA instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0111 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
then
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
&& IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
                AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Any, TLBI_ExcludeXS, R[t]);
            else
                AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL2 then
        AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH, TLBILevel_Any,
TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL3 then
        AArch32.TLBI_VAA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_NSH,
TLBILevel_Any, TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAAIS, TLB Invalidate by VA, All ASID, Inner Shareable

The TLBIMVAAIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from any level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIMVAAIS are UNDEFINED.

Attributes

TLBIMVAAIS is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA | | | | | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | |

VA, bits [31:12]

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVAAIS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0011 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
        && IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBI_Level_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBI_Level_Any, TLBI_AllAttr, R[t]);
        endif
    endif
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH, TLBI_Level_Any,
    TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    AArch32.TLBI_VAA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_ISH,
    TLBI_Level_Any, TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAAL, TLB Invalidate by VA, All ASID, Last level

The TLBIMVAAL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIMVAAL are UNDEFINED.

Note

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVAAL is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA | | | | | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | |

VA, bits [31:12]

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVAAL instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0111 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
        then
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Last, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Last, TLBI_AllAttr, R[t]);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Last, TLBI_AllAttr, R[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
            && IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
                AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Last, TLBI_ExcludeXS, R[t]);
            else
                AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Last, TLBI_AllAttr, R[t]);
            elsif PSTATE.EL == EL2 then
                AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
                TLBIlevel_Last, TLBI_AllAttr, R[t]);
            elsif PSTATE.EL == EL3 then
                AArch32.TLBI_VAA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_NSH,
                TLBIlevel_Last, TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAALIS, TLB Invalidate by VA, All ASID, Last level, Inner Shareable

The TLBIMVAALIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry, from the final level of the translation table walk.
- The entry would be used to translate the specified address.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIMVAALIS are UNDEFINED.

Note

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVAALIS is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA | | | | | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | |

VA, bits [31:12]

Virtual address to match. Any unlocked TLB entries that match the VA will be affected by this System instruction, regardless of the ASID.

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVAALIS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0011 | 0b111 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
        && IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Last, TLBI_ExcldeXS, R[t]);
        else
            AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Last, TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL2 then
        AArch32.TLBI_VAA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
        TLBIlevel_Last, TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL3 then
        AArch32.TLBI_VAA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_ISH,
        TLBIlevel_Last, TLBI_AllAttr, R[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAH, TLB Invalidate by VA, Hyp mode

The TLBIMVAH characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIMVAH are UNDEFINED.

Attributes

TLBIMVAH is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA | | | | | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | |

VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVAH instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1000 | 0b0111 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Shareability_NSH,
    TLBIlevel_Any, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_VA(SS_NonSecure, Regime_EL2, VMID[], Shareability_NSH, TLBIlevel_Any,
        TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAHIS, TLB Invalidate by VA, Hyp mode, Inner Shareable

The TLBIMVAHIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from any level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

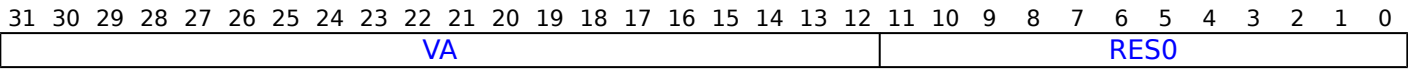
Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIMVAHIS are UNDEFINED.

Attributes

TLBIMVAHIS is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVAHIS instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1000 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Shareability_ISH,
    TLBIlevel_Any, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_VA(SS_NonSecure, Regime_EL2, VMID[], Shareability_ISH, TLBIlevel_Any,
        TLBI_AllAttr, R[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAIS, TLB Invalidate by VA, Inner Shareable

The TLBIMVAIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
 - The entry is from a level of lookup above the final level and matches the specified ASID.
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIMVAIS are UNDEFINED.

Attributes

TLBIMVAIS is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA | | | | | | | | | | | | RES0 | | | | ASID | | | | | | | | | | | | | | | |

VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

Executing the TLBIMVAIS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0011 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
        && IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBILevel_Any, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBILevel_Any, TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL2 then
        AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH, TLBILevel_Any,
        TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL3 then
        AArch32.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_ISH,
        TLBILevel_Any, TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVAL, TLB Invalidate by VA, Last level

The TLBIMVAL characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation only applies to the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIMVAL are UNDEFINED.

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVAL is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA | | | | | | | | | | | | RES0 | | | | ASID | | | | | | | | | | | | | | | |

VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

Executing the TLBIMVAL instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0111 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.FB == '1' then
        if IsFeatureImplemented(FEAT_XS) && IsFeatureImplemented(FEAT_HCX) && HCRX_EL2.FnXS == '1'
then
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, R[t]);
        elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.FB == '1' then
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
TLBILevel_Last, TLBI_AllAttr, R[t]);
        else
            if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
&& IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
                AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_ExcludeXS, R[t]);
            else
                AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL2 then
        AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_NSH, TLBILevel_Last,
TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL3 then
        AArch32.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_NSH,
TLBILevel_Last, TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVALH, TLB Invalidate by VA, Last level, Hyp mode

The TLBIMVALH characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from the final level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation only applies to the PE that executes this System instruction.

Configuration

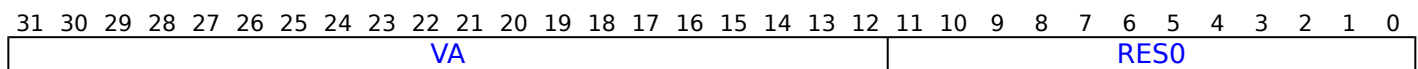
This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIMVALH are UNDEFINED.

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVALH is a 32-bit System instruction.

Field descriptions



VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVALH instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1000 | 0b0111 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Shareability_NSH,
    TLBIlevel_Last, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_VA(SS_NonSecure, Regime_EL2, VMID[], Shareability_NSH, TLBIlevel_Last,
        TLBI_AllAttr, R[t]);

```

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVALHIS, TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable

The TLBIMVALHIS characteristics are:

Purpose

If EL2 is implemented, invalidate all cached copies of translation table entries from TLBs that are from the final level of the translation table walk that would be required for the Non-secure EL2 translation regime and used to translate the specified address.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIMVALHIS are UNDEFINED.

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVALHIS is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA | | | | | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | |

VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:0]

Reserved, RES0.

Executing the TLBIMVALHIS instruction

If this instruction is executed in a Secure privileged mode other than Monitor mode, then the behavior is CONSTRAINED UNPREDICTABLE, and one of the following behaviors must occur:

- The instruction is UNDEFINED.
- The instruction is treated as a NOP.
- The instruction executes as if it had been executed in Monitor mode.

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1000 | 0b0011 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    AArch32.TLBI_VA(SecurityStateAtEL(EL2), Regime_EL2, VMID_NONE, Shareability_ISH,
    TLBIlevel_Last, TLBI_AllAttr, R[t]);
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        UNDEFINED;
    else
        AArch32.TLBI_VA(SS_NonSecure, Regime_EL2, VMID[], Shareability_ISH, TLBIlevel_Last,
        TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBIMVALIS, TLB Invalidate by VA, Last level, Inner Shareable

The TLBIMVALIS characteristics are:

Purpose

Invalidate all cached copies of translation table entries from TLBs that meet the following requirements:

- The entry is a stage 1 translation table entry.
- The entry would be used to translate the specified address, and one of the following applies:
 - The entry is a global entry from the final level of lookup.
 - The entry is a non-global entry from the final level of lookup that matches the specified ASID.
- If EL2 is implemented and enabled in the current Security state, the entry would be used with the current VMID.

From the entries that match these requirements, the entries that are invalidated are required for the following translation regime:

- If executed at Secure EL1 when EL3 is using AArch64, the Secure EL1&0 translation regime.
- If executed in Secure state when EL3 is using AArch32, the Secure PL1&0 translation regime.
- If executed in Non-secure state, the Non-secure PL1&0 translation regime.

The invalidation applies to all PEs in the same Inner Shareable shareability domain as the PE that executes this System instruction.

Configuration

This instruction is present only when AArch32 is supported. Otherwise, direct accesses to TLBIMVALIS are UNDEFINED.

This System instruction is not implemented in architecture versions before Armv8.

Attributes

TLBIMVALIS is a 32-bit System instruction.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VA | | | | | | | | | | | RES0 | | | | ASID | | | | | | | | | | | | | | | | |

VA, bits [31:12]

Virtual address to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Bits [11:8]

Reserved, RES0.

ASID, bits [7:0]

ASID value to match. Any TLB entries that match the ASID value and VA value will be affected by this System instruction.

Global TLB entries that match the VA value will be affected by this System instruction, regardless of the value of the ASID field.

Executing the TLBIMVALIS instruction

Accesses to this instruction use the following encodings in the System instruction encoding space:

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1000 | 0b0011 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T8 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T8 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLB == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TTLBIS == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TTLB == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR2.TTLBIS == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        if IsFeatureImplemented(FEAT_XS) && !ELUsingAArch32(EL2) && IsFeatureImplemented(FEAT_HCX)
        && IsHCRXEL2Enabled() && HCRX_EL2.FnXS == '1' then
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Last, TLBI_ExcludeXS, R[t]);
        else
            AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH,
            TLBIlevel_Last, TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL2 then
        AArch32.TLBI_VA(SecurityStateAtEL(EL1), Regime_EL10, VMID[], Shareability_ISH, TLBIlevel_Last,
        TLBI_AllAttr, R[t]);
    elsif PSTATE.EL == EL3 then
        AArch32.TLBI_VA(SecurityStateAtEL(EL3), Regime_EL30, VMID_NONE, Shareability_ISH,
        TLBIlevel_Last, TLBI_AllAttr, R[t]);

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TLBTR, TLB Type Register

The TLBTR characteristics are:

Purpose

Provides information about the TLB implementation. The register must define whether the implementation provides separate instruction and data TLBs, or a unified TLB. Normally, the IMPLEMENTATION DEFINED information in this register includes the number of lockable entries in the TLB.

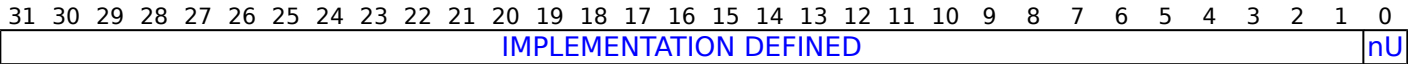
Configuration

This register is present only when AArch32 is supported. Otherwise, direct accesses to TLBTR are UNDEFINED.

Attributes

TLBTR is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:1]

IMPLEMENTATION DEFINED.

nU, bit [0]

Not Unified TLB. Indicates whether the implementation has a unified TLB:

| nU | Meaning |
|-----|-------------------------------------|
| 0b0 | Unified TLB. |
| 0b1 | Separate Instruction and Data TLBs. |

Accessing TLBTR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TID1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TID1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        return TLBTR;
elsif PSTATE.EL == EL2 then
    return TLBTR;
elsif PSTATE.EL == EL3 then
    return TLBTR;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDRPRW, PL1 Software Thread ID Register

The TPIDRPRW characteristics are:

Purpose

Provides a location where software executing at EL1 or higher can store thread identifying information that is not visible to software executing at EL0, for OS management purposes.

The PE makes no use of this register.

Configuration

AArch32 System register TPIDRPRW bits [31:0] are architecturally mapped to AArch64 System register [TPIDR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to TPIDRPRW are UNDEFINED.

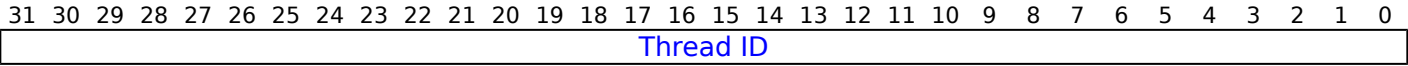
Note

The PE never updates this register.

Attributes

TPIDRPRW is a 32-bit register.

Field descriptions



Bits [31:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDRPRW

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TPIDRPRW_NS;
    else
        return TPIDRPRW;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TPIDRPRW_NS;
    else
        return TPIDRPRW;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TPIDRPRW_S;
    else
        return TPIDRPRW_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b100 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        TPIDRPRW_NS = R[t];
    else
        TPIDRPRW = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TPIDRPRW_NS = R[t];
    else
        TPIDRPRW = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        TPIDRPRW_S = R[t];
    else
        TPIDRPRW_NS = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TPIDRURO, PL0 Read-Only Software Thread ID Register

The TPIDRURO characteristics are:

Purpose

Provides a location where software executing at EL1 or higher can store thread identifying information that is visible to software executing at EL0, for OS management purposes.

The PE makes no use of this register.

Configuration

AArch32 System register TPIDRURO bits [31:0] are architecturally mapped to AArch64 System register [TPIDRRO_EL0\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to TPIDRURO are UNDEFINED.

Note

The PE never updates this register.

Attributes

TPIDRURO is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Thread ID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDRURO

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b011 |

```

if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGTR_EL2.TPIDRRO_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        return TPIDRURO;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TPIDRURO_NS;
    else
        return TPIDRURO;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TPIDRURO_NS;
    else
        return TPIDRURO;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TPIDRURO_S;
    else
        return TPIDRURO_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        TPIDRURO_NS = R[t];
    else
        TPIDRURO = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TPIDRURO_NS = R[t];
    else
        TPIDRURO = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        TPIDRURO_S = R[t];
    else
        TPIDRURO_NS = R[t];

```

TPIDRURW, PL0 Read/Write Software Thread ID Register

The TPIDRURW characteristics are:

Purpose

Provides a location where software executing at EL0 can store thread identifying information, for OS management purposes.

The PE makes no use of this register.

Configuration

AArch32 System register TPIDRURW bits [31:0] are architecturally mapped to AArch64 System register [TPIDR_EL0\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to TPIDRURW are UNDEFINED.

Note

The PE never updates this register.

Attributes

TPIDRURW is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Thread ID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Thread ID. Thread identifying information stored by software running at this Exception level.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing TPIDRURW

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGTR_EL2.TPIDR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        return TPIDRURW;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TPIDRURW_NS;
    else
        return TPIDRURW;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TPIDRURW_NS;
    else
        return TPIDRURW;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TPIDRURW_S;
    else
        return TPIDRURW_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.<E2H,TGE> != '11' && HSTR_EL2.T13 == '1'
    then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL1) && HCR_EL2.<E2H,TGE> != '11' && (!HaveEL(EL3) ||
SCR_EL3.FGTEn == '1') && HFGWTR_EL2.TPIDR_EL0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    else
        TPIDRURW = R[t];
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T13 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T13 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        TPIDRURW_NS = R[t];
    else
        TPIDRURW = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TPIDRURW_NS = R[t];
    else
        TPIDRURW = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        TPIDRURW_S = R[t];
    else
        TPIDRURW_NS = R[t];

```


TRFCR, Trace Filter Control Register

The TRFCR characteristics are:

Purpose

Provides EL1 controls for Trace.

Configuration

AArch32 System register TRFCR bits [31:0] are architecturally mapped to AArch64 System register [TRFCR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported and FEAT_TRF is implemented. Otherwise, direct accesses to TRFCR are UNDEFINED.

Attributes

TRFCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|----|---|------|---|-------|---|-------|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | TS | | RES0 | | E1TRE | | E0TRE | |

Bits [31:7]

Reserved, RES0.

TS, bits [6:5]

Timestamp Control. Controls which timebase is used for trace timestamps.

| TS | Meaning | Applies when |
|------|--|------------------------------|
| 0b01 | Virtual timestamp. The traced timestamp is the physical counter value minus the value of CNTVOFF . | |
| 0b10 | Guest physical timestamp. The traced timestamp is the physical counter value minus a physical offset. If any of the following are true, the physical offset is zero, otherwise the physical offset is the value of CNTPOFF_EL2 : <ul style="list-style-type: none"> EL3 is implemented and is using AArch32. EL3 is implemented, using AArch64, and SCR_EL3.ECVEn == 0b0. EL2 is using AArch32. EL2 is using AArch64 and CNTHCTL_EL2.ECV == 0b0. | When FEAT_ECV is implemented |
| 0b11 | Physical timestamp. The traced timestamp is the physical counter value. | |

All other values are reserved.

This field is ignored by the PE when any of the following are true:

- EL2 is implemented and [HTRFCR.TS](#) != 0b00.
- SelfHostedTraceEnabled() == FALSE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [4:2]

Reserved, RES0.

E1TRE, bit [1]

EL1 Trace Enable.

| E1TRE | Meaning |
|-------|-------------------------------------|
| 0b0 | Tracing is prohibited in PL1 modes. |
| 0b1 | Tracing is allowed in PL1 modes. |

This field is ignored if SelfHostedTraceEnabled() == FALSE.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

E0TRE, bit [0]

EL0 Trace Enable.

| E0TRE | Meaning |
|-------|-------------------------------|
| 0b0 | Tracing is prohibited at EL0. |
| 0b1 | Tracing is allowed at EL0. |

This field is ignored if any of the following are true:

- SelfHostedTraceEnabled() == FALSE.
- EL2 is implemented and enabled in the current security state and [HCR](#).TGE == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing TRFCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SDCR.TTRF == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TTRF == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TTRF == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SDCR.TTRF == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return TRFCR;
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            return TRFCR;
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SDCR.TTRF == '1' then
            AArch32.TakeMonitorTrapException();
        else
            return TRFCR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0001 | 0b0010 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        UNDEFINED;
    elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SDCR.TTRF == '1'
then
        UNDEFINED;
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T1 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T1 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && MDCR_EL2.TTRF == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HDCR.TTRF == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
        if Halted() && EDSCR.SDD == '1' then
            UNDEFINED;
        else
            AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && PSTATE.M != M32_Monitor && SDCR.TTRF == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            TRFCR = R[t];
    elsif PSTATE.EL == EL2 then
        if Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
            UNDEFINED;
        elsif Halted() && HaveEL(EL3) && EDSCR.SDD == '1' && boolean IMPLEMENTATION_DEFINED "EL3 trap
priority when SDD == '1'" && ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
            UNDEFINED;
        elsif HaveEL(EL3) && !ELUsingAArch32(EL3) && MDCR_EL3.TTRF == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch64.AArch32SystemAccessTrap(EL3, 0x03);
        elsif HaveEL(EL3) && ELUsingAArch32(EL3) && SDCR.TTRF == '1' then
            if Halted() && EDSCR.SDD == '1' then
                UNDEFINED;
            else
                AArch32.TakeMonitorTrapException();
        else
            TRFCR = R[t];
    elsif PSTATE.EL == EL3 then
        if PSTATE.M != M32_Monitor && SDCR.TTRF == '1' then
            AArch32.TakeMonitorTrapException();
        else
            TRFCR = R[t];

```

TTBCR, Translation Table Base Control Register

The TTBCR characteristics are:

Purpose

The control register for stage 1 of the PL1&0 translation regime. Its controls include:

- Where the VA range is split between addresses translated using [TTBR0](#) and addresses translated using [TTBR1](#).
- The translation table format used by this stage of translation.

From Armv8.2, when the value of TTBCR.{EAE, T2E} is {1, 1}, TTBCR is used with [TTBCR2](#).

Configuration

AArch32 System register TTBCR bits [31:0] are architecturally mapped to AArch64 System register [TCR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to TTBCR are UNDEFINED.

The current translation table format determines which format of the register is used.

Some RW fields of this register have defined reset values. These apply only if the PE resets into an Exception level that is using AArch32. If the PE resets into EL3 using AArch32 then:

- The EAE bit resets to 0 in both the Secure and the Non-secure instances of the register.
- Other reset values apply only to the Secure instance of the register.

Attributes

TTBCR is a 32-bit register.

Field descriptions

When TTBCR.EAE == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EAE | | RES0 | | | | | | | | | | | | | | | | | | PD1PD0 | | RES0 | | N | | | | | | | |

EAE, bit [31]

Extended Address Enable.

| EAE | Meaning |
|-----|--|
| 0b0 | Use the VMSAv8-32 translation system with the Short-descriptor translation table format. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [30:6]

Reserved, RES0.

PD1, bit [5]

Translation table walk disable for translations using [TTBR1](#). This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using [TTBR1](#).

| PD1 | Meaning |
|-----|--|
| 0b0 | Perform translation table walks using TTBR1 . |
| 0b1 | A TLB miss on an address that is translated using TTBR1 generates a Translation fault. No translation table walk is performed. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

PD0, bit [4]

Translation table walk disable for translations using [TTBR0](#). This bit controls whether a translation table walk is performed on a TLB miss for an address that is translated using [TTBR0](#).

| PD0 | Meaning |
|-----|--|
| 0b0 | Perform translation table walks using TTBR0 . |
| 0b1 | A TLB miss on an address that is translated using TTBR0 generates a Translation fault. No translation table walk is performed. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bit [3]

Reserved, RES0.

N, bits [2:0]

Indicate the width of the base address held in [TTBR0](#). In [TTBR0](#), the base address field is bits[31:14-N]. The value of N also determines:

- Whether [TTBR0](#) or [TTBR1](#) is used as the base address for translation table walks.
- The size of the translation table pointed to by [TTBR0](#).

N can take any value from 0 to 7, that is, from 0b000 to 0b111.

When N has its reset value of 0, the translation table base is compatible with Armv5 and Armv6.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

When TTBCR.EAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|---------------------------|----|-----|-------|-------|------|----|------|------|------|-----|-------|-------|------|------|------|------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EAE | IMPLEMENTATION DEFINED | | SH1 | ORGN1 | IRGN1 | EPD1 | A1 | RES0 | T1SZ | RES0 | SH0 | ORGN0 | IRGN0 | EPD0 | T2SZ | RES0 | T0SZ | | | | | | | | | | | | | | |

EAE, bit [31]

Extended Address Enable.

| EAE | Meaning |
|-----|---|
| 0b1 | Use the VMSAv8-32 translation system with the Long-descriptor translation table format. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

IMPLEMENTATION DEFINED, bit [30]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

SH1, bits [29:28]

Shareability attribute for memory associated with translation table walks using [TTBR1](#).

| SH1 | Meaning |
|------|------------------|
| 0b00 | Non-shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

ORGN1, bits [27:26]

Outer cacheability attribute for memory associated with translation table walks using [TTBR1](#).

| ORGN1 | Meaning |
|-------|---|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

IRGN1, bits [25:24]

Inner cacheability attribute for memory associated with translation table walks using [TTBR1](#).

| IRGN1 | Meaning |
|-------|---|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

EPD1, bit [23]

Translation table walk disable for translations using [TTBR1](#). This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using [TTBR1](#).

| EPD1 | Meaning |
|------|--|
| 0b0 | Perform translation table walks using TTBR1 . |
| 0b1 | A TLB miss on an address that is translated using TTBR1 generates a Translation fault. No translation table walk is performed. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

A1, bit [22]

Selects whether [TTBR0](#) or [TTBR1](#) defines the ASID.

| A1 | Meaning |
|-----|---|
| 0b0 | TTBR0 .ASID defines the ASID. |
| 0b1 | TTBR1 .ASID defines the ASID. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [21:19]

Reserved, RES0.

T1SZ, bits [18:16]

See 'Selecting between TTBR0 and TTBR1, VMSAv8-32 Long-descriptor translation table format' for how TTBCR.{T1SZ, T0SZ} determine the input address ranges and memory region sizes translated using [TTBR0](#) and [TTBR1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [15:14]

Reserved, RES0.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using [TTBR0](#).

| SH0 | Meaning |
|------|-----------------|
| 0b00 | Non-shareable |
| 0b10 | Outer Shareable |
| 0b11 | Inner Shareable |

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using [TTBR0](#).

| ORGNO | Meaning |
|-------|---|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using [TTBR0](#).

| IRGN0 | Meaning |
|-------|---|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

EPD0, bit [7]

Translation table walk disable for translations using [TTBR0](#). This bit controls whether a translation table walk is performed on a TLB miss, for an address that is translated using [TTBR0](#).

| EPD0 | Meaning |
|------|--|
| 0b0 | Perform translation table walks using TTBR0 . |
| 0b1 | A TLB miss on an address that is translated using TTBR0 generates a Translation fault. No translation table walk is performed. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

T2E, bit [6]

When FEAT_AA32HPD is implemented:

TTBCR2 Enable.

| T2E | Meaning |
|-----|--|
| 0b0 | TTBCR2 is disabled. The contents of TTBCR2 are treated as 0 for all purposes other than reading or writing the register. |
| 0b1 | TTBCR2 is enabled. |

If TTBCR.EAE==0, then the behavior is as if the bit is 0.

Otherwise:

Reserved, RAZ/WI.

Bits [5:3]

Reserved, RES0.

T0SZ, bits [2:0]

See 'Selecting between TTBR0 and TTBR1, VMSAv8-32 Long-descriptor translation table format' for how TTBCR.{T1SZ, T0SZ} determine the input address ranges and memory region sizes translated using [TTBR0](#) and [TTBR1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing TTBCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0010 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBCR_NS;
    else
        return TTBCR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBCR_NS;
    else
        return TTBCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TTBCR_S;
    else
        return TTBCR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0010 | 0b0000 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBCR_NS = R[t];
    else
        TTBCR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBCR_NS = R[t];
    else
        TTBCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBCR_S = R[t];
        else
            TTBCR_NS = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBCR2, Translation Table Base Control Register 2

The TTBCR2 characteristics are:

Purpose

The second control register for stage 1 of the PL1&0 translation regime.

If FEAT_AA32HPD is not implemented then this register is not implemented and its encoding is UNDEFINED. Otherwise:

- When the value of [TTBCR](#).{EAE, T2E} is not {1, 1} the contents of TTBCR2 are treated as zero for all purposes other than reading or writing the register.
- When the value of [TTBCR](#).{EAE, T2E} is {1, 1} TTBCR2 is used with [TTBCR](#).

Configuration

AArch32 System register TTBCR2 bits [31:0] are architecturally mapped to AArch64 System register [TCR_EL1\[63:32\]](#).

This register is present only when AArch32 is supported and FEAT_AA32HPD is implemented. Otherwise, direct accesses to TTBCR2 are UNDEFINED.

Attributes

TTBCR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|--------|--------|--------|--------|--------|--------|--------|--------|------|------|----|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | HWU162 | HWU161 | HWU160 | HWU159 | HWU062 | HWU061 | HWU060 | HWU059 | HPD1 | HPD0 | RE | | | | | | | | |

Bits [31:19]

Reserved, RES0.

HWU162, bit [18] When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using [TTBR1](#).

| HWU162 | Meaning |
|--------|--|
| 0b0 | For translations using TTBR1 , bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR1 , bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1. |

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of [TTBCR](#).T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU161, bit [17]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using [TTBR1](#).

| HWU161 | Meaning |
|--------|--|
| 0b0 | For translations using TTBR1 , bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR1 , bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1. |

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of [TTBCR](#).T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU160, bit [16]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using [TTBR1](#).

| HWU160 | Meaning |
|--------|--|
| 0b0 | For translations using TTBR1 , bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR1 , bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1. |

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of [TTBCR](#).T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU159, bit [15]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using [TTBR1](#).

| HWU159 | Meaning |
|--------|--|
| 0b0 | For translations using TTBR1 , bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR1 , bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD1 is 1. |

The Effective value of this field is 0 if the value of TTBCR2.HPD1 is 0 or the value of [TTBCR](#).T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU062, bit [14]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 1 translation table Block or Page entry for translations using [TTBR0](#).

| HWU062 | Meaning |
|--------|--|
| 0b0 | For translations using TTBR0 , bit[62] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR0 , bit[62] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1. |

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of [TTBCR](#).T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU061, bit [13]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 1 translation table Block or Page entry for translations using [TTBR0](#).

| HWU061 | Meaning |
|--------|--|
| 0b0 | For translations using TTBR0 , bit[61] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR0 , bit[61] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1. |

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of [TTBCR](#).T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU060, bit [12]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 1 translation table Block or Page entry for translations using [TTBR0](#).

| HWU060 | Meaning |
|--------|--|
| 0b0 | For translations using TTBR0 , bit[60] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR0 , bit[60] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1. |

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of [TTBCR](#).T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU059, bit [11]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 1 translation table Block or Page entry for translations using [TTBR0](#).

| HWU059 | Meaning |
|--------|--|
| 0b0 | For translations using TTBR0 , bit[59] of each stage 1 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | For translations using TTBR0 , bit[59] of each stage 1 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose if the value of TTBCR2.HPD0 is 1. |

The Effective value of this field is 0 if the value of TTBCR2.HPD0 is 0 or the value of [TTBCR](#).T2E is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HPD1, bit [10]

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, XNTable, and PXNTable, in the translation tables pointed to by [TTBR1](#).

| HPD1 | Meaning |
|------|---|
| 0b0 | Hierarchical permissions are enabled. |
| 0b1 | Hierarchical permissions are disabled if TTBCR .T2E == 1. |

When disabled, the permissions are treated as if the bits are 0.

The Effective value of this field is 0 if the value of [TTBCR.T2E](#) is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

HPD0, bit [9]

Hierarchical Permission Disables. This affects the hierarchical control bits, APTable, XNTable, and PXNTable, in the translation tables pointed to by [TTBR0](#).

| HPD0 | Meaning |
|------|--|
| 0b0 | Hierarchical permissions are enabled. |
| 0b1 | Hierarchical permissions are disabled if TTBCR.T2E == 1. |

When disabled, the permissions are treated is as if the bits are 0.

The Effective value of this field is 0 if the value of [TTBCR.T2E](#) is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:0]

Reserved, RES0.

Accessing TTBCR2

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0010 | 0b0000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBCR2_NS;
    else
        return TTBCR2;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBCR2_NS;
    else
        return TTBCR2;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TTBCR2_S;
    else
        return TTBCR2_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0010 | 0b0000 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBCR2_NS = R[t];
    else
        TTBCR2 = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBCR2_NS = R[t];
    else
        TTBCR2 = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBCR2_S = R[t];
        else
            TTBCR2_NS = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TTBR0, Translation Table Base Register 0

The TTBR0 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the lower VA range in the PL1&0 translation regime, and other information for this translation regime.

Configuration

AArch32 System register TTBR0 bits [63:0] are architecturally mapped to AArch64 System register [TTBR0_EL1\[63:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to TTBR0 are UNDEFINED.

TTBR0 is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits [31:0] and do not modify bits [63:32].

[TTBCR](#).EAE determines which TTBR0 format is used:

- [TTBCR](#).EAE == 0b0: 32-bit format is used. TTBR0[63:32] are ignored.
- [TTBCR](#).EAE == 0b1: 64-bit format is used.

When EL3 is using AArch32, write access to TTBR0(S) is disabled when the CP15SDISABLE signal is asserted HIGH.

Used in conjunction with the [TTBCR](#). When the 64-bit TTBR0 format is used, cacheability and shareability information is held in the [TTBCR](#), not in TTBR0.

Attributes

TTBR0 is a 64-bit register.

Field descriptions

When TTBCR.EAE == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|-----|----|-----|----|-----|----|---|--|---------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TTB0 | | | | | | | | | | | | | | | | | | | | | | | | IRGN[0] | | NOS | | RGN | | IMP | | S | | IRGN[1] | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

Bits [63:32]

Reserved, RES0.

TTB0, bits [31:7]

Translation table base address, bits[31:x], where x is 14-(TTBCR.N). Register bits [x-1:7] are RES0, with the additional requirement that if these bits are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:7] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN, bits [0, 6]

Inner region bits. Bits [0,6] of this register together indicate the Inner Cacheability attributes for the memory associated with the translation table walks. The possible values of IRGN[1:0] are:

| IRGN | Meaning |
|------|--|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Cacheable. |
| 0b11 | Normal memory, Inner Write-Back no Write-Allocate Cacheable. |

Note

The encoding of the IRGN bits is counter-intuitive, with register bit[6] being IRGN[0] and register bit[0] being IRGN[1]. This encoding is chosen to give a consistent encoding of memory region types and to ensure that software written for ARMv7 without the Multiprocessing Extensions can run unmodified on an implementation that includes the functionality introduced by the ARMv7 Multiprocessing Extensions.

The IRGN field is split as follows:

- IRGN[0] is TTBR0[6].
- IRGN[1] is TTBR0[0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NOS, bit [5]

Not Outer Shareable. When the value of TTBR0.S is 1, indicates whether the memory associated with a translation table walk is Inner Shareable or Outer Shareable:

| NOS | Meaning |
|-----|----------------------------|
| 0b0 | Memory is Outer Shareable. |
| 0b1 | Memory is Inner Shareable. |

This bit is ignored when the value of TTBR0.S is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RGN, bits [4:3]

Region bits. Indicates the Outer cacheability attributes for the memory associated with the translation table walks:

| RGN | Meaning |
|------|--|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Cacheable. |
| 0b11 | Normal memory, Outer Write-Back no Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMP, bit [2]

The effect of this bit is IMPLEMENTATION DEFINED. If the translation table implementation does not include any IMPLEMENTATION DEFINED features this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [1]

Shareable. Indicates whether the memory associated with the translation table walks is Shareable:

| S | Meaning |
|-----|--|
| 0b0 | Memory is Non-shareable. |
| 0b1 | Memory is Shareable. The TTBR0.NOS field indicates whether the memory is Inner Shareable or Outer Shareable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When TTBCR.EAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | ASID | | | | | | | | BADDR | | | | | | | | | | | | | | | |
| BADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | CnP |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:56]

Reserved, RES0.

ASID, bits [55:48]

An ASID for the translation table base address. The [TTBCR.A1](#) field selects either TTBR0.ASID or [TTBR1.ASID](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR, bits [47:1]

Translation table base address, bits[47:x], Bits [x-1:1] are RES0, with the additional requirement that if bits[x-1:3] are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:3] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

x is determined from the value of [TTBCR.T0SZ](#) as follows:

- If [TTBCR.T0SZ](#) is 0 or 1, $x = 5 - \text{TTBCR.T0SZ}$.
- If [TTBCR.T0SZ](#) is greater than 1, $x = 14 - \text{TTBCR.T0SZ}$.

If bits[47:40] of the translation table base address are not zero, an Address size fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. When [TTBCR.EAE](#) ==1, this bit indicates whether each entry that is pointed to by TTBR0 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR0.CnP is 1.

| CnP | Meaning |
|-----|--|
| 0b0 | <p>The translation table entries pointed to by this instance of TTBR0, for the current ASID, are permitted to differ from corresponding entries for this instance of TTBR0 for other PEs in the Inner Shareable domain. This is not affected by:</p> <ul style="list-style-type: none"> • The value of TTBR0.CnP on those other PEs. • The value of TTBCR.EAE on those other PEs. • The value of the current ASID or, for the Non-secure instance of TTBR0, the value of the current VMID. |
| 0b1 | <p>The translation table entries pointed to by this instance of TTBR0 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR0.CnP is 1 for this instance of TTBR0 and all of the following apply:</p> <ul style="list-style-type: none"> • The translation table entries are pointed to by this instance of TTBR0. • The value of the applicable TTBCR.EAE field is 1. • The ASID is the same as the current ASID. • For the Non-secure instance of TTBR0, the VMID is the same as the current VMID. |

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR0.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR0s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are **CONSTRAINED UNPREDICTABLE**, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR0

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0010 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBR0_NS<31:0>;
    else
        return TTBR0<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBR0_NS<31:0>;
    else
        return TTBR0<31:0>;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TTBR0_S<31:0>;
    else
        return TTBR0_NS<31:0>;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0010 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR0_NS = ZeroExtend(R[t]);
    else
        TTBR0 = ZeroExtend(R[t]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR0_NS = ZeroExtend(R[t]);
    else
        TTBR0 = ZeroExtend(R[t]);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR0_S = ZeroExtend(R[t]);
        else
            TTBR0_NS = ZeroExtend(R[t]);

```

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b0010 | 0b0000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBR0_NS;
    else
        return TTBR0;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBR0_NS;
    else
        return TTBR0;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TTBR0_S;
    else
        return TTBR0_NS;

```

MCCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b0010 | 0b0000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR0_NS = R[t2]:R[t];
    else
        TTBR0 = R[t2]:R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR0_NS = R[t2]:R[t];
    else
        TTBR0 = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR0_S = R[t2]:R[t];
        else
            TTBR0_NS = R[t2]:R[t];

```

TTBR1, Translation Table Base Register 1

The TTBR1 characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 1 of the translation of an address from the higher VA range in the PL1&0 translation regime, and other information for this translation regime.

Configuration

AArch32 System register TTBR1 bits [63:0] are architecturally mapped to AArch64 System register [TTBR1_EL1\[63:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to TTBR1 are UNDEFINED.

TTBR1 is a 64-bit register that can also be accessed as a 32-bit value. If it is accessed as a 32-bit register, accesses read and write bits [31:0] and do not modify bits [63:32].

[TTBCR](#).EAE determines which TTBR1 format is used:

- [TTBCR](#).EAE == 0b0: 32-bit format is used. TTBR1[63:32] are ignored.
- [TTBCR](#).EAE == 0b1: 64-bit format is used.

Used in conjunction with the [TTBCR](#). When the 64-bit TTBR1 format is used, cacheability and shareability information is held in the [TTBCR](#), not in TTBR1.

Attributes

TTBR1 is a 64-bit register.

Field descriptions

When TTBCR.EAE == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|-----|-----|----|-----|----|---------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| TTB1 | | | | | | | | | | | | | | | | | | | | | | | | IRGN[1] | | NOS | RGN | | IMP | S | IRGN[0] | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:32]

Reserved, RES0.

TTB1, bits [31:7]

Translation table base address, bits[31:14]. Register bits [13:7] are RES0, with the additional requirement that if these bits are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [13:7] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN, bits [6, 0]

Inner region bits. IRGN[1:0] indicate the Inner Cacheability attributes for the memory associated with the translation table walks. The possible values of IRGN[1:0] are:

| IRGN | Meaning |
|------|--|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Cacheable. |
| 0b11 | Normal memory, Inner Write-Back no Write-Allocate Cacheable. |

Note

The encoding of the IRGN bits is counter-intuitive, with register bit[6] being IRGN[0] and register bit[0] being IRGN[1]. This encoding is chosen to give a consistent encoding of memory region types and to ensure that software written for Armv7 without the Multiprocessing Extensions can run unmodified on an implementation that includes the functionality introduced by the ARMv7 Multiprocessing Extensions.

The IRGN field is split as follows:

- IRGN[1] is TTBR1[6].
- IRGN[0] is TTBR1[0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NOS, bit [5]

Not Outer Shareable. When the value of TTBR1.S is 1, indicates whether the memory associated with a translation table walk is Inner Shareable or Outer Shareable:

| NOS | Meaning |
|-----|----------------------------|
| 0b0 | Memory is Outer Shareable. |
| 0b1 | Memory is Inner Shareable. |

This bit is ignored when the value of TTBR1.S is 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

RGN, bits [4:3]

Region bits. Indicates the Outer cacheability attributes for the memory associated with the translation table walks:

| RGN | Meaning |
|------|--|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Cacheable. |
| 0b11 | Normal memory, Outer Write-Back no Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IMP, bit [2]

The effect of this bit is IMPLEMENTATION DEFINED. If the translation table implementation does not include any IMPLEMENTATION DEFINED features this bit is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

S, bit [1]

Shareable. Indicates whether the memory associated with the translation table walks is Shareable:

| S | Meaning |
|-----|--|
| 0b0 | Memory is Non-shareable. |
| 0b1 | Memory is Shareable. The TTBR1.NOS field indicates whether the memory is Inner Shareable or Outer Shareable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

When TTBCR.EAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | ASID | | | | | | | | BADDR | | | | | | | | | | | | | | | | |
| BADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | CnP | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:56]

Reserved, RES0.

ASID, bits [55:48]

An ASID for the translation table base address. The [TTBCR.A1](#) field selects either [TTBR0.ASID](#) or [TTBR1.ASID](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

BADDR, bits [47:1]

Translation table base address, bits[47:x], Bits [x-1:1] are RES0, with the additional requirement that if bits[x-1:3] are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:3] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

x is determined from the value of [TTBCR.T1SZ](#) as follows:

- If [TTBCR.T1SZ](#) is 0 or 1, $x = 5 - \text{TTBCR.T1SZ}$.
- If [TTBCR.T1SZ](#) is greater than 1, $x = 14 - \text{TTBCR.T1SZ}$.

If bits[47:40] of the translation table base address are not zero, an Address size fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. When [TTBCR.EAE](#) == 1, this bit indicates whether each entry that is pointed to by TTBR1 is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of TTBR1.CnP is 1.

| CnP | Meaning |
|-----|--|
| 0b0 | <p>The translation table entries pointed to by this instance of TTBR1, for the current ASID, are permitted to differ from corresponding entries for this instance of TTBR1 for other PEs in the Inner Shareable domain. This is not affected by:</p> <ul style="list-style-type: none"> • The value of TTBR1.CnP on those other PEs. • The value of TTBCR.EAE on those other PEs. • The value of the current ASID or, for the Non-secure instance of TTBR1, the value of the current VMID. |
| 0b1 | <p>The translation table entries pointed to by this instance of TTBR1 are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of TTBR1.CnP is 1 for this instance of TTBR1 and all of the following apply:</p> <ul style="list-style-type: none"> • The translation table entries are pointed to by this instance of TTBR1. • The value of the applicable TTBCR.EAE field is 1. • The ASID is the same as the current ASID. • For the Non-secure instance of TTBR1, the VMID is the same as the current VMID. |

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the TTBR1.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those TTBR1s do not point to the same translation table entries when the other conditions specified for the case when the value of CnP is 1 apply, then the results of translations are **CONSTRAINED UNPREDICTABLE**, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TTBR1

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0010 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBR1_NS<31:0>;
    else
        return TTBR1<31:0>;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBR1_NS<31:0>;
    else
        return TTBR1<31:0>;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TTBR1_S<31:0>;
    else
        return TTBR1_NS<31:0>;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0010 | 0b0000 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR1_NS = ZeroExtend(R[t]);
    else
        TTBR1 = ZeroExtend(R[t]);
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR1_NS = ZeroExtend(R[t]);
    else
        TTBR1 = ZeroExtend(R[t]);
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR1_S = ZeroExtend(R[t]);
        else
            TTBR1_NS = ZeroExtend(R[t]);

```

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b0010 | 0b0001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TRVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TRVM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBR1_NS;
    else
        return TTBR1;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return TTBR1_NS;
    else
        return TTBR1;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return TTBR1_S;
    else
        return TTBR1_NS;

```

MCCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b0010 | 0b0001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.TVM == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HCR.TVM == '1' then
        AArch32.TakeHypTrapException(0x04);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR1_NS = R[t2]:R[t];
    else
        TTBR1 = R[t2]:R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        TTBR1_NS = R[t2]:R[t];
    else
        TTBR1 = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            TTBR1_S = R[t2]:R[t];
        else
            TTBR1_NS = R[t2]:R[t];

```

VBAR, Vector Base Address Register

The VBAR characteristics are:

Purpose

When high exception vectors are not selected, holds the vector base address for exceptions that are not taken to Monitor mode or to Hyp mode.

Software must program VBAR(NS) with the required initial value as part of the PE boot sequence.

Configuration

AArch32 System register VBAR bits [31:0] are architecturally mapped to AArch64 System register [VBAR_EL1\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to VBAR are UNDEFINED.

Attributes

VBAR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|------|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Vector Base Address | | | | | | | | | | | | | | | | | | | | | | | | | | RES0 | | | | | |

Bits [31:5]

Vector Base Address. Bits[31:5] of the base address of the exception vectors for exceptions taken to this Exception level. Bits[4:0] of an exception vector are the exception offset.

The reset behavior of this field is:

- On a Warm reset, this field resets to an IMPLEMENTATION DEFINED value.

Bits [4:0]

Reserved, RES0.

Accessing VBAR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        return VBAR_NS;
    else
        return VBAR;
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        return VBAR_NS;
    else
        return VBAR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        return VBAR_S;
    else
        return VBAR_NS;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif HaveEL(EL3) && ELUsingAArch32(EL3) then
        VBAR_NS = R[t];
    else
        VBAR = R[t];
elsif PSTATE.EL == EL2 then
    if HaveEL(EL3) && ELUsingAArch32(EL3) then
        VBAR_NS = R[t];
    else
        VBAR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' && CP15SDISABLE == HIGH then
        UNDEFINED;
    elsif SCR.NS == '0' && CP15SDISABLE2 == HIGH then
        UNDEFINED;
    else
        if SCR.NS == '0' then
            VBAR_S = R[t];
        else
            VBAR_NS = R[t];

```

VDFSR, Virtual SError Exception Syndrome Register

The VDFSR characteristics are:

Purpose

Provides the syndrome value reported to software on taking a virtual SError interrupt exception to EL1, or on executing an ESB instruction at EL1.

When the virtual SError interrupt injected using [HCR.VA](#) is taken to EL1 using AArch32, then the syndrome value is reported in [DFSR](#).{AET, ExT} and the remainder of [DFSR](#) is set as defined by VMSAv8-32. For more information, see The AArch32 Virtual Memory System Architecture.

If the virtual SError interrupt injected using [HCR.VA](#) is deferred by an ESB instruction, then the syndrome value is written to [VDISR](#).

Configuration

AArch32 System register VDFSR bits [31:0] are architecturally mapped to AArch64 System register [VSESR_EL2\[31:0\]](#) when the highest implemented Exception level is using AArch64.

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to VDFSR are UNDEFINED.

If EL2 is not implemented, then VDFSR is RES0 from Monitor mode when [SCR.NS](#) == 1.

Attributes

VDFSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|---------|----|------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | AET | | RES0ExT | | RES0 | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

AET, bits [15:14]

When a virtual SError interrupt is taken to EL1 using AArch32, [DFSR](#)[15:4] is set to VDFSR.AET.

When a virtual SError interrupt is deferred by an ESB instruction, [VDISR](#)[15:4] is set to VDFSR.AET.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

When a virtual SError interrupt is taken to EL1 using AArch32, [DFSR](#)[12] is set to VDFSR.ExT.

When a virtual SError interrupt is deferred by an ESB instruction, [VDISR](#)[12] is set to VDFSR.ExT.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:0]

Reserved, RES0.

Accessing VDFSR

Direct reads and writes of VDFSR are UNDEFINED if EL3 is implemented and using AArch32 in all Secure privileged modes other than Monitor mode.

If EL2 is not implemented, then VDFSR is RES0 from Monitor mode when [SCR.NS](#) == 1.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0101 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VDFSR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return VDFSR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0101 | 0b0010 | 0b011 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T5 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T5 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VDFSR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        VDFSR = R[t];

```


VDISR, Virtual Deferred Interrupt Status Register

The VDISR characteristics are:

Purpose

Records that an SError interrupt has been consumed by an ESB instruction.

Configuration

AArch32 System register VDISR bits [31:0] are architecturally mapped to AArch64 System register [VDISR_EL2\[31:0\]](#).

This register is present only when FEAT_RAS is implemented. Otherwise, direct accesses to VDISR are UNDEFINED.

If EL2 is not implemented, then VDISR is RES0 from Monitor mode when SCR.NS == 1.

Attributes

VDISR is a 32-bit register.

Field descriptions

When TTBCR.EAE == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|----|----|----|----|----|----|-----|------|-----|------|-------|------|------|----|----|----|----|---------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| A | RES0 | | | | | | | | | | AET | RES0 | ExT | RES0 | FS[4] | LPAE | RES0 | | | | | FS[3:0] | | | | | | | | | |

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

The value copied from [VDFSR.AET](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

The value copied from [VDFSR.ExT](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [11]

Reserved, RES0.

FS, bits [10, 3:0]

Fault status code. Set to 0b10110 when an ESB instruction defers a virtual SError interrupt.

| FS | Meaning |
|---------|--------------------------------|
| 0b10110 | Asynchronous SError interrupt. |

All other values are reserved.

The FS field is split as follows:

- FS[4] is VDISR[10].
- FS[3:0] is VDISR[3:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LPAE, bit [9]

Format.

Set to [TTBCR.EAE](#) when an ESB instruction defers a virtual SError interrupt.

| LPAE | Meaning |
|------|--|
| 0b0 | Using the Short-descriptor translation table format. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:4]

Reserved, RES0.

When TTBCR.EAE == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|------|----|----|----|----|----|----|----|----|----|-----|------|-----|------|------|------|--------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| A | RES0 | | | | | | | | | | AET | RES0 | EXT | RES0 | LPAE | RES0 | STATUS | | | | | | | | | | | | | | |

A, bit [31]

Set to 1 when an ESB instruction defers a virtual SError interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [30:16]

Reserved, RES0.

AET, bits [15:14]

The value copied from [VDESR.AET](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [13]

Reserved, RES0.

ExT, bit [12]

The value copied from [VDFSR.ExT](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [11:10]

Reserved, RES0.

LPAE, bit [9]

Format.

Set to [TTBCR.EAE](#) when an ESB instruction defers a virtual SError interrupt.

| LPAE | Meaning |
|------|---|
| 0b1 | Using the Long-descriptor translation table format. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:6]

Reserved, RES0.

STATUS, bits [5:0]

Fault status code. Set to 0b010001 when an ESB instruction defers a virtual SError interrupt.

| STATUS | Meaning |
|----------|--------------------------------|
| 0b010001 | Asynchronous SError interrupt. |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VDISR

Direct reads and writes of VDFSR are UNDEFINED if EL3 is implemented and using AArch32 in all Secure privileged modes other than Monitor mode.

An indirect write to VDISR made by an ESB instruction does not require an explicit synchronization operation for the value that is written to be observed by a direct read of [DISR](#) occurring in program order after the ESB instruction.

If EL2 is not implemented, then VDISR is RES0 from Monitor mode when [SCR.NS](#) == 1.

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VDISR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return VDISR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b1100 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VDISR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        VDISR = R[t];

```

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
        return VDISR_EL2;
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.AMO == '1' then
        return VDISR;
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return Zeros();
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && !Halted() && SCR.EA == '1' then
        return Zeros();
    else
        return DISR;
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && !Halted() && SCR_EL3.EA == '1' then
        return Zeros();
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && !Halted() && SCR.EA == '1' then
        return Zeros();
    else
        return DISR;
elseif PSTATE.EL == EL3 then
    return DISR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b1100 | 0b0001 | 0b001 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elseif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T12 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T12 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elseif EL2Enabled() && !ELUsingAArch32(EL2) && HCR_EL2.AMO == '1' then
        VDISR_EL2 = R[t];
    elseif EL2Enabled() && ELUsingAArch32(EL2) && HCR.AMO == '1' then
        VDISR = R[t];
    elseif HaveEL(EL3) && !ELUsingAArch32(EL3) && !Halted() && SCR_EL3.EA == '1' then
        //no operation
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && !Halted() && SCR.EA == '1' then
        //no operation
    else
        DISR = R[t];
elseif PSTATE.EL == EL2 then
    if HaveEL(EL3) && !ELUsingAArch32(EL3) && !Halted() && SCR_EL3.EA == '1' then
        //no operation
    elseif HaveEL(EL3) && ELUsingAArch32(EL3) && !Halted() && SCR.EA == '1' then
        //no operation
    else
        DISR = R[t];
elseif PSTATE.EL == EL3 then
    DISR = R[t];

```

VMPIDR, Virtualization Multiprocessor ID Register

The VMPIDR characteristics are:

Purpose

Holds the value of the Virtualization Multiprocessor ID. This is the value returned by Non-secure EL1 reads of [MPIDR](#).

Configuration

AArch32 System register VMPIDR bits [31:0] are architecturally mapped to AArch64 System register [VMPIDR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to VMPIDR are UNDEFINED.

If EL2 is not implemented but EL3 is implemented, this register takes the value of the [MPIDR](#).

Attributes

VMPIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|------|----|----|----|----|----|------|----|----|----|----|------|----|----|----|----|------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| M | U | RES0 | | | | | MT | Aff2 | | | | | Aff1 | | | | | Aff0 | | | | | | | | | | | | | |

M, bit [31]

Indicates whether this implementation includes the functionality introduced by the Armv7 Multiprocessing Extensions.

| M | Meaning |
|-----|--|
| 0b0 | This implementation does not include the Armv7 Multiprocessing Extensions functionality. |
| 0b1 | This implementation includes the Armv7 Multiprocessing Extensions functionality. |

Access to this field is **RES1**.

U, bit [30]

Indicates a Uniprocessor system, as distinct from PE 0 in a multiprocessor system.

| U | Meaning |
|-----|---|
| 0b0 | Processor is part of a multiprocessor system. |
| 0b1 | Processor is part of a uniprocessor system. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to the value in [MPIDR.U](#).

Bits [29:25]

Reserved, RES0.

MT, bit [24]

Indicates whether the lowest level of affinity consists of logical PEs that are implemented using a multithreading type approach. See the description of Aff0 for more information about affinity levels.

| MT | Meaning |
|-----|---|
| 0b0 | Performance of PEs at the lowest affinity level is largely independent. |
| 0b1 | Performance of PEs at the lowest affinity level is very interdependent. |

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to the value in [MPIDR.MT](#).

Aff2, bits [23:16]

Affinity level 2. See the description of Aff0 for more information.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to the value in [MPIDR.Aff2](#).

Aff1, bits [15:8]

Affinity level 1. See the description of Aff0 for more information.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to the value in [MPIDR.Aff1](#).

Aff0, bits [7:0]

Affinity level 0. This is the affinity level that is most significant for determining PE behavior. Higher affinity levels are increasingly less significant in determining PE behavior. The assigned value of the [MPIDR_EL1](#).{Aff3, Aff2, Aff1, Aff0} set of fields of each PE must be unique within the system as a whole.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to the value in [MPIDR.Aff0](#).

Accessing VMPIDR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0000 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VMPIDR;
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        return MPIDR;
    elsif SCR.NS == '0' then
        UNDEFINED;
    else
        return VMPIDR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0000 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VMPIDR = R[t];
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        //no operation
    elsif SCR.NS == '0' then
        UNDEFINED;
    else
        VMPIDR = R[t];

```

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b101 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) then
        return VMPIDR_EL2<31:0>;
    elsif EL2Enabled() && ELUsingAArch32(EL2) then
        return VMPIDR;
    else
        return MPIDR;
elsif PSTATE.EL == EL2 then
    return MPIDR;
elsif PSTATE.EL == EL3 then
    return MPIDR;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VPIDR, Virtualization Processor ID Register

The VPIDR characteristics are:

Purpose

Holds the value of the Virtualization Processor ID. This is the value returned by Non-secure EL1 reads of [MIDR](#).

Configuration

AArch32 System register VPIDR bits [31:0] are architecturally mapped to AArch64 System register [VPIDR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to VPIDR are UNDEFINED.

If EL2 is not implemented but EL3 is implemented, this register takes the value of the [MIDR](#).

Attributes

VPIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|---------|----|----|----|--------------|----|----|----|---------|----|----|----|----|----|---|---|----------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Implementer | | | | | | | | Variant | | | | Architecture | | | | PartNum | | | | | | | | Revision | | | | | | | |

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

| Implementer | Meaning |
|-------------|--|
| 0x00 | Reserved for software use. |
| 0x41 | Arm Limited. |
| 0x42 | Broadcom Corporation. |
| 0x43 | Cavium Inc. |
| 0x44 | Digital Equipment Corporation. |
| 0x46 | Fujitsu Ltd. |
| 0x49 | Infineon Technologies AG. |
| 0x4D | Motorola or Freescale Semiconductor Inc. |
| 0x4E | NVIDIA Corporation. |
| 0x50 | Applied Micro Circuits Corporation. |
| 0x51 | Qualcomm Inc. |
| 0x56 | Marvell International Ltd. |
| 0x69 | Intel Corporation. |
| 0xC0 | Ampere Computing. |

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to the value in [MIDR](#).Implementer.

Variant, bits [23:20]

An IMPLEMENTATION DEFINED variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to the value in [MIDR.Variant](#).

Architecture, bits [19:16]

Architecture version. Defined values are:

| Architecture | Meaning |
|--------------|---|
| 0b0001 | Armv4. |
| 0b0010 | Armv4T. |
| 0b0011 | Armv5 (obsolete). |
| 0b0100 | Armv5T. |
| 0b0101 | Armv5TE. |
| 0b0110 | Armv5TEJ. |
| 0b0111 | Armv6. |
| 0b1111 | Architectural features are individually identified in the ID_* registers. |

All other values are reserved.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to the value in [MIDR.Architecture](#).

PartNum, bits [15:4]

An IMPLEMENTATION DEFINED primary part number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to the value in [MIDR.PartNum](#).

Revision, bits [3:0]

An IMPLEMENTATION DEFINED revision number for the device.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to the value in [MIDR.Revision](#).

Accessing VPIDR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0000 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VPIDR;
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        return MIDR;
    elsif SCR.NS == '0' then
        UNDEFINED;
    else
        return VPIDR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0000 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VPIDR = R[t];
elsif PSTATE.EL == EL3 then
    if !HaveEL(EL2) then
        //no operation
    elsif SCR.NS == '0' then
        UNDEFINED;
    else
        VPIDR = R[t];

```

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b000 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T0 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T0 == '1' then
        AArch32.TakeHypTrapException(0x03);
    elsif EL2Enabled() && !ELUsingAArch32(EL2) then
        return VPIDR_EL2<31:0>;
    elsif EL2Enabled() && ELUsingAArch32(EL2) then
        return VPIDR;
    else
        return MIDR;
elsif PSTATE.EL == EL2 then
    return MIDR;
elsif PSTATE.EL == EL3 then
    return MIDR;

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VTCR, Virtualization Translation Control Register

The VTCR characteristics are:

Purpose

The control register for stage 2 of the Non-secure PL1&0 translation regime.

Note

This stage of translation always uses the Long-descriptor translation table format.

Configuration

AArch32 System register VTCR bits [31:0] are architecturally mapped to AArch64 System register [VTCR_EL2\[31:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to VTCR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

VTCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|-------|-------|-------|-------|------|----|----|----|----|-----|-------|-------|-----|------|----|------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES1 | RES0 | HWU62 | HWU61 | HWU60 | HWU59 | RES0 | | | | | SH0 | ORGN0 | IRGN0 | SLO | RES0 | S | TOSZ | | | | | | | | | | | | | | |

Bit [31]

Reserved, RES1.

Bits [30:29]

Reserved, RES0.

HWU62, bit [28]

When FEAT_HPDS2 is implemented:

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[62] of the stage 2 translation table Block or Page entry.

| HWU62 | Meaning |
|-------|---|
| 0b0 | Bit[62] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[62] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU61, bit [27]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[61] of the stage 2 translation table Block or Page entry.

| HWU61 | Meaning |
|-------|---|
| 0b0 | Bit[61] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[61] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU60, bit [26]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[60] of the stage 2 translation table Block or Page entry.

| HWU60 | Meaning |
|-------|---|
| 0b0 | Bit[60] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[60] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

HWU59, bit [25]**When FEAT_HPDS2 is implemented:**

Hardware Use. Indicates IMPLEMENTATION DEFINED hardware use of bit[59] of the stage 2 translation table Block or Page entry.

| HWU59 | Meaning |
|-------|---|
| 0b0 | Bit[59] of each stage 2 translation table Block or Page entry cannot be used by hardware for an IMPLEMENTATION DEFINED purpose. |
| 0b1 | Bit[59] of each stage 2 translation table Block or Page entry can be used by hardware for an IMPLEMENTATION DEFINED purpose. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RAZ/WI.

Bits [24:14]

Reserved, RES0.

SH0, bits [13:12]

Shareability attribute for memory associated with translation table walks using [VTTBR](#).

| SH0 | Meaning |
|------|------------------|
| 0b00 | Non-shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

Other values are reserved. The effect of programming this field to a Reserved value is that behavior is CONSTRAINED UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

ORGN0, bits [11:10]

Outer cacheability attribute for memory associated with translation table walks using [VTTBR](#).

| ORGN0 | Meaning |
|-------|---|
| 0b00 | Normal memory, Outer Non-cacheable. |
| 0b01 | Normal memory, Outer Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Outer Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Outer Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

IRGN0, bits [9:8]

Inner cacheability attribute for memory associated with translation table walks using [VTTBR](#).

| IRGN0 | Meaning |
|-------|---|
| 0b00 | Normal memory, Inner Non-cacheable. |
| 0b01 | Normal memory, Inner Write-Back Read-Allocate Write-Allocate Cacheable. |
| 0b10 | Normal memory, Inner Write-Through Read-Allocate No Write-Allocate Cacheable. |
| 0b11 | Normal memory, Inner Write-Back Read-Allocate No Write-Allocate Cacheable. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

SL0, bits [7:6]

Starting level for translation table walks using [VTTBR](#).

| SL0 | Meaning |
|------|------------------|
| 0b00 | Start at level 2 |
| 0b01 | Start at level 1 |

All other values are reserved. If this field is programmed to a reserved value, or to a value that is not consistent with the programming of T0SZ, then a stage 2 level 1 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bit [5]

Reserved, RES0.

S, bit [4]

Sign extension bit. This bit must be programmed to the value of T0SZ[3]. If it is not, then the stage 2 T0SZ value is treated as an UNKNOWN value

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

T0SZ, bits [3:0]

The size offset of the memory region addressed by [VTTBR](#). The region size is $2^{(32-T0SZ)}$ bytes.

This field holds a four-bit signed integer value, meaning it supports values from -8 to 7.

Note

This is different from the other translation control registers, where TnSZ holds a three-bit unsigned integer, supporting values from 0 to 7.

If this field is programmed to a value that is not consistent with the programming of SL0 then a stage 2 level 1 Translation fault is generated.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing VTCR

Accesses to this register use the following encodings in the System register encoding space:

MRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0010 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VTCR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return VTCR;

```

MCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <CRn>, <CRm>{, {#}<opc2>}

| coproc | opc1 | CRn | CRm | opc2 |
|--------|-------|--------|--------|-------|
| 0b1111 | 0b100 | 0b0010 | 0b0001 | 0b010 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x03);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x03);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTCR = R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        VTCR = R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

VTTBR, Virtualization Translation Table Base Register

The VTTBR characteristics are:

Purpose

Holds the base address of the translation table for the initial lookup for stage 2 of an address translation in the Non-secure PL1&0 translation regime, and other information for this translation regime.

Configuration

AArch32 System register VTTBR bits [63:0] are architecturally mapped to AArch64 System register [VTTBR_EL2\[63:0\]](#).

This register is present only when AArch32 is supported. Otherwise, direct accesses to VTTBR are UNDEFINED.

If EL2 is not implemented, this register is RES0 from EL3.

Attributes

VTTBR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | VMID | | | | | | | | BADDR | | | | | | | | | | | | | | | |
| BADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | CnP |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:56]

Reserved, RES0.

VMID, bits [55:48]

The VMID for the translation table.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to 0.

BADDR, bits [47:1]

Translation table base address, bits[47:x], Bits [x-1:1] are RES0, with the additional requirement that if bits[x-1:3] are not all zero, this is a misaligned translation table base address, with effects that are CONSTRAINED UNPREDICTABLE, and must be one of the following:

- Register bits [x-1:3] are treated as if all the bits are zero. The value read back from these bits is either the value written or zero.
- The result of the calculation of an address for a translation table walk using this register can be corrupted in those bits that are nonzero.

x is determined from the value of [VTCR.SL0](#) and [VTCR.T0SZ](#) as follows:

- If [VTCR.SL0](#) is 0b00, meaning that lookup starts at level 2, then x is 14 - [VTCR.T0SZ](#).
- If [VTCR.SL0](#) is 0b01, meaning that lookup starts at level 1, then x is 5 - [VTCR.T0SZ](#).
- If [VTCR.SL0](#) is either 0b10 or 0b11 then a stage 2 level 1 Translation fault is generated.

If bits[47:40] of the translation table base address are not zero, an Address size fault is generated.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to an architecturally UNKNOWN value.

CnP, bit [0]

When FEAT_TTCNP is implemented:

Common not Private. This bit indicates whether each entry that is pointed to by VTTBR is a member of a common set that can be used by every PE in the Inner Shareable domain for which the value of VTTBR.CnP is 1.

| CnP | Meaning |
|-----|--|
| 0b0 | The translation table entries pointed to by VTTBR are permitted to differ from the entries for VTTBR for other PEs in the Inner Shareable domain. This is not affected by the value of the current VMID. |
| 0b1 | The translation table entries pointed to by VTTBR are the same as the translation table entries for every other PE in the Inner Shareable domain for which the value of VTTBR.CnP is 1 and the VMID is the same as the current VMID. |

When a TLB combines entries from stage 1 translation and stage 2 translation into a single entry, that entry can only be shared between different PEs if the value of the CnP bit is 1 for both stage 1 and stage 2.

Note

If the value of the VTTBR.CnP bit is 1 on multiple PEs in the same Inner Shareable domain and those VTTBRs do not point to the same translation table entries when the VMID value is the same as the current VMID, then the results of translations are CONSTRAINED UNPREDICTABLE, see 'CONSTRAINED UNPREDICTABLE behaviors due to caching of control or data values'.

The reset behavior of this field is:

- On a Warm reset, in a system where the PE resets into EL2 or EL3, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing VTTBR

Accesses to this register use the following encodings in the System register encoding space:

MRRC{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b0010 | 0b0110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    return VTTBR;
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        return VTTBR;

```

MCCR{<c>}{<q>} <coproc>, {#}<opc1>, <Rt>, <Rt2>, <CRm>

| coproc | CRm | opc1 |
|--------|--------|--------|
| 0b1111 | 0b0010 | 0b0110 |

```

if PSTATE.EL == EL0 then
    UNDEFINED;
elsif PSTATE.EL == EL1 then
    if EL2Enabled() && !ELUsingAArch32(EL2) && HSTR_EL2.T2 == '1' then
        AArch64.AArch32SystemAccessTrap(EL2, 0x04);
    elsif EL2Enabled() && ELUsingAArch32(EL2) && HSTR.T2 == '1' then
        AArch32.TakeHypTrapException(0x04);
    else
        UNDEFINED;
elsif PSTATE.EL == EL2 then
    VTTBR = R[t2]:R[t];
elsif PSTATE.EL == EL3 then
    if SCR.NS == '0' then
        UNDEFINED;
    else
        VTTBR = R[t2]:R[t];

```

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

System Register index by instruction and encoding

Below are indexes for registers and operations accessed in the following ways:

For AArch32

- [MCR/MRC](#)
- [MCRR/MRRC](#)
- [MRS/MSR](#)
- [VMRS/VMSR](#)

For AArch64

- [AT](#)
- [BRB](#)
- [CFP](#)
- [CPP](#)
- [DC](#)
- [DVP](#)
- [IC](#)
- [MRS/MSR](#)
- [TLBI](#)

Registers and operations in AArch32

Accessed using MCR/MRC:

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|-------|-----------------------------|---|
| | | CRn | CRm | | | |
| 0b1110 | 0b000 | 0b0000 | 0b0000 | 0b000 | DBGDIDR | Debug ID Register |
| 0b1110 | 0b000 | 0b0000 | 0b0000 | 0b010 | DBGDTRRXext | Debug OS Local Data Transfer Register, Receive, External View |
| 0b1110 | 0b000 | 0b0000 | 0b0001 | 0b000 | DBGDSCRint | Debug Status and Control Register, Internal View |
| 0b1110 | 0b000 | 0b0000 | 0b0010 | 0b000 | DBGDCCINT | DCC Interrupt Enable Register |
| 0b1110 | 0b000 | 0b0000 | 0b0010 | 0b010 | DBGDSCRext | Debug Status and Control Register, External View |
| 0b1110 | 0b000 | 0b0000 | 0b0011 | 0b010 | DBGDTRTXext | Debug OS Local Data Transfer Register, Transmit |
| 0b1110 | 0b000 | 0b0000 | 0b0101 | 0b000 | DBGDTRRXint | Debug Data Transfer Register, Receive |
| 0b1110 | 0b000 | 0b0000 | 0b0101 | 0b000 | DBGDTRTXint | Debug Data Transfer Register, Transmit |
| 0b1110 | 0b000 | 0b0000 | 0b0110 | 0b000 | DBGWFEAR | Debug Watchpoint Fault Address Register |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|-------|----------------------------------|--|
| | | CRn | CRm | | | |
| 0b1110 | 0b000 | 0b0000 | 0b0110 | 0b010 | DBGOSECCR | Debug OS Lock Exception Catch Control Register |
| 0b1110 | 0b000 | 0b0000 | 0b0111 | 0b000 | DBGVCR | Debug Vector Catch Register |
| 0b1110 | 0b000 | 0b0000 | n[3:0] | 0b100 | DBGBVR<n> | Debug Breakpoint Value Register |
| 0b1110 | 0b000 | 0b0000 | n[3:0] | 0b101 | DBGBCR<n> | Debug Breakpoint Control Registers |
| 0b1110 | 0b000 | 0b0000 | n[3:0] | 0b110 | DBGWVR<n> | Debug Watchpoint Value Register |
| 0b1110 | 0b000 | 0b0000 | n[3:0] | 0b111 | DBGWCR<n> | Debug Watchpoint Control Registers |
| 0b1110 | 0b000 | 0b0001 | 0b0000 | 0b000 | DBGDRAR | Debug ROM Address Register |
| 0b1110 | 0b000 | 0b0001 | 0b0000 | 0b100 | DBGOSLAR | Debug OS Lock Access Register |
| 0b1110 | 0b000 | 0b0001 | 0b0001 | 0b100 | DBGOSLSR | Debug OS Lock Status Register |
| 0b1110 | 0b000 | 0b0001 | 0b0011 | 0b100 | DBGOSDLR | Debug OS Double Lock Register |
| 0b1110 | 0b000 | 0b0001 | 0b0100 | 0b100 | DBGPRCR | Debug Power Control Register |
| 0b1110 | 0b000 | 0b0001 | n[3:0] | 0b001 | DBGBXVR<n> | Debug Breakpoint Extended Value Registers |
| 0b1110 | 0b000 | 0b0010 | 0b0000 | 0b000 | DBGDSAR | Debug Self Address Register |
| 0b1110 | 0b000 | 0b0111 | 0b0000 | 0b111 | DBGDEVID2 | Debug Device ID register 2 |
| 0b1110 | 0b000 | 0b0111 | 0b0001 | 0b111 | DBGDEVID1 | Debug Device ID register 1 |
| 0b1110 | 0b000 | 0b0111 | 0b0010 | 0b111 | DBGDEVID | Debug Device ID register 0 |
| 0b1110 | 0b000 | 0b0111 | 0b1000 | 0b110 | DBGCLAIMSET | Debug CLAIM Tag Set register |
| 0b1110 | 0b000 | 0b0111 | 0b1001 | 0b110 | DBGCLAIMCLR | Debug CLAIM Tag Clear register |
| 0b1110 | 0b000 | 0b0111 | 0b1110 | 0b110 | DBGAUTHSTATUS | Debug Authentication Status register |
| 0b1110 | 0b111 | 0b0000 | 0b0000 | 0b000 | JIDR | Jazelle ID Register |
| 0b1110 | 0b111 | 0b0001 | 0b0000 | 0b000 | JOSCR | Jazelle OS Control Register |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|-------|--------------------------|--------------------------------------|
| | | CRn | CRm | | | |
| 0b1110 | 0b111 | 0b0010 | 0b0000 | 0b000 | JMCR | Jazelle Main Configuration Register |
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b000 | MIDR | Main ID Register |
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b001 | CTR | Cache Type Register |
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b010 | TCMTR | TCM Type Register |
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b011 | TLBTR | TLB Type Register |
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b101 | MPIDR | Multiprocessor Affinity Register |
| 0b1111 | 0b000 | 0b0000 | 0b0000 | 0b110 | REVIDR | Revision ID Register |
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b000 | ID_PFR0 | Processor Feature Register 0 |
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b001 | ID_PFR1 | Processor Feature Register 1 |
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b010 | ID_DFR0 | Debug Feature Register 0 |
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b011 | ID_AFR0 | Auxiliary Feature Register 0 |
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b100 | ID_MMFR0 | Memory Model Feature Register 0 |
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b101 | ID_MMFR1 | Memory Model Feature Register 1 |
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b110 | ID_MMFR2 | Memory Model Feature Register 2 |
| 0b1111 | 0b000 | 0b0000 | 0b0001 | 0b111 | ID_MMFR3 | Memory Model Feature Register 3 |
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b000 | ID_ISAR0 | Instruction Set Attribute Register 0 |
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b001 | ID_ISAR1 | Instruction Set Attribute Register 1 |
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b010 | ID_ISAR2 | Instruction Set Attribute Register 2 |
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b011 | ID_ISAR3 | Instruction Set Attribute Register 3 |
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b100 | ID_ISAR4 | Instruction Set Attribute Register 4 |
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b101 | ID_ISAR5 | Instruction Set Attribute Register 5 |
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b110 | ID_MMFR4 | Memory Model Feature Register 4 |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|-------|--------------------------|---|
| | | CRn | CRm | | | |
| 0b1111 | 0b000 | 0b0000 | 0b0010 | 0b111 | ID_ISAR6 | Instruction Set Attribute Register 6 |
| 0b1111 | 0b000 | 0b0000 | 0b0011 | 0b100 | ID_PFR2 | Processor Feature Register 2 |
| 0b1111 | 0b000 | 0b0000 | 0b0011 | 0b101 | ID_DFR1 | Debug Feature Register 1 |
| 0b1111 | 0b000 | 0b0000 | 0b0011 | 0b110 | ID_MMFR5 | Memory Model Feature Register 5 |
| 0b1111 | 0b000 | 0b0001 | 0b0000 | 0b000 | SCTLR | System Control Register |
| 0b1111 | 0b000 | 0b0001 | 0b0000 | 0b001 | ACTLR | Auxiliary Control Register |
| 0b1111 | 0b000 | 0b0001 | 0b0000 | 0b010 | CPACR | Architectural Feature Access Control Register |
| 0b1111 | 0b000 | 0b0001 | 0b0000 | 0b011 | ACTLR2 | Auxiliary Control Register 2 |
| 0b1111 | 0b000 | 0b0001 | 0b0001 | 0b000 | SCR | Secure Configuration Register |
| 0b1111 | 0b000 | 0b0001 | 0b0001 | 0b001 | SDER | Secure Debug Enable Register |
| 0b1111 | 0b000 | 0b0001 | 0b0001 | 0b010 | NSACR | Non-Secure Access Control Register |
| 0b1111 | 0b000 | 0b0001 | 0b0010 | 0b001 | TRFCR | Trace Filter Control Register |
| 0b1111 | 0b000 | 0b0001 | 0b0011 | 0b001 | SDCR | Secure Debug Control Register |
| 0b1111 | 0b000 | 0b0010 | 0b0000 | 0b000 | TTBR0 | Translation Table Base Register 0 |
| 0b1111 | 0b000 | 0b0010 | 0b0000 | 0b001 | TTBR1 | Translation Table Base Register 1 |
| 0b1111 | 0b000 | 0b0010 | 0b0000 | 0b010 | TTBCR | Translation Table Base Control Register |
| 0b1111 | 0b000 | 0b0010 | 0b0000 | 0b011 | TTBCR2 | Translation Table Base Control Register 2 |
| 0b1111 | 0b000 | 0b0011 | 0b0000 | 0b000 | DACR | Domain Access Control Register |
| 0b1111 | 0b000 | 0b0100 | 0b0110 | 0b000 | ICC_PMR | Interrupt Controller Interrupt Priority Mask Register |
| 0b1111 | 0b000 | 0b0101 | 0b0000 | 0b000 | DFSR | Data Fault Status Register |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|-------|---------------------------|--|
| | | CRn | CRm | | | |
| 0b1111 | 0b000 | 0b0101 | 0b0000 | 0b001 | IFSR | Instruction Fault Status Register |
| 0b1111 | 0b000 | 0b0101 | 0b0001 | 0b000 | ADFSR | Auxiliary Data Fault Status Register |
| 0b1111 | 0b000 | 0b0101 | 0b0001 | 0b001 | AIFSR | Auxiliary Instruction Fault Status Register |
| 0b1111 | 0b000 | 0b0101 | 0b0011 | 0b000 | ERRIDR | Error Record Register |
| 0b1111 | 0b000 | 0b0101 | 0b0011 | 0b001 | ERRSELR | Error Record Select Register |
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b000 | ERXFR | Selected Error Record Feature Register |
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b001 | ERXCTLR | Selected Error Record Control Register |
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b010 | ERXSTATUS | Selected Error Record Primary Status Register |
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b011 | ERXADDR | Selected Error Record Address Register |
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b100 | ERXFR2 | Selected Error Record Feature Register 2 |
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b101 | ERXCTLR2 | Selected Error Record Control Register 2 |
| 0b1111 | 0b000 | 0b0101 | 0b0100 | 0b111 | ERXADDR2 | Selected Error Record Address Register 2 |
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b000 | ERXMISC0 | Selected Error Record Miscellaneous Register 0 |
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b001 | ERXMISC1 | Selected Error Record Miscellaneous Register 1 |
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b010 | ERXMISC4 | Selected Error Record Miscellaneous Register 4 |
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b011 | ERXMISC5 | Selected Error Record Miscellaneous Register 5 |
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b100 | ERXMISC2 | Selected Error Record Miscellaneous Register 2 |
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b101 | ERXMISC3 | Selected Error Record Miscellaneous Register 3 |
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b110 | ERXMISC6 | Selected Error Record |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|-------|--------------------------|--|
| | | CRn | CRm | | | |
| | | | | | | Miscellaneous Register 6 |
| 0b1111 | 0b000 | 0b0101 | 0b0101 | 0b111 | ERXMISC7 | Selected Error Record Miscellaneous Register 7 |
| 0b1111 | 0b000 | 0b0110 | 0b0000 | 0b000 | DFAR | Data Fault Address Register |
| 0b1111 | 0b000 | 0b0110 | 0b0000 | 0b010 | IFAR | Instruction Fault Address Register |
| 0b1111 | 0b000 | 0b0111 | 0b0001 | 0b000 | ICIALUIS | Instruction Cache Invalidate All PoU, Inner Shareable |
| 0b1111 | 0b000 | 0b0111 | 0b0001 | 0b110 | BPIALLIS | Branch Predictor Invalidate All, Inner Shareable |
| 0b1111 | 0b000 | 0b0111 | 0b0011 | 0b100 | CFPRCTX | Control Flow Prediction Restriction by Context |
| 0b1111 | 0b000 | 0b0111 | 0b0011 | 0b101 | DVPRCTX | Data Value Prediction Restriction by Context |
| 0b1111 | 0b000 | 0b0111 | 0b0011 | 0b111 | CPPRCTX | Cache Prefetch Prediction Restriction by Context |
| 0b1111 | 0b000 | 0b0111 | 0b0100 | 0b000 | PAR | Physical Address Register |
| 0b1111 | 0b000 | 0b0111 | 0b0101 | 0b000 | ICIALLU | Instruction Cache Invalidate All PoU |
| 0b1111 | 0b000 | 0b0111 | 0b0101 | 0b001 | ICIMVAU | Instruction Cache line Invalidate by VA to PoU |
| 0b1111 | 0b000 | 0b0111 | 0b0101 | 0b100 | CP15ISB | Instruction Synchronization Barrier System instruction |
| 0b1111 | 0b000 | 0b0111 | 0b0101 | 0b110 | BPIALL | Branch Predictor Invalidate All |
| 0b1111 | 0b000 | 0b0111 | 0b0101 | 0b111 | BPIMVA | Branch Predictor Invalidate by VA |
| 0b1111 | 0b000 | 0b0111 | 0b0110 | 0b001 | DCIMVAC | Data Cache line Invalidate by VA to PoC |
| 0b1111 | 0b000 | 0b0111 | 0b0110 | 0b010 | DCISW | Data Cache line Invalidate by Set/Way |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|-------|----------------------------|---|
| | | CRn | CRm | | | |
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b000 | ATS1CPR | Address Translate Stage 1 Current stage PL1 Read |
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b001 | ATS1CPW | Address Translate Stage 1 Current stage PL1 Write |
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b010 | ATS1CUR | Address Translate Stage 1 Current stage Unprivileged Read |
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b011 | ATS1CUW | Address Translate Stage 1 Current stage Unprivileged Write |
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b100 | ATS12NSOPR | Address Translate Stages 1 and Non-secure Only PL1 Read |
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b101 | ATS12NSOPW | Address Translate Stages 1 and Non-secure Only PL1 Write |
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b110 | ATS12NSOUR | Address Translate Stages 1 and Non-secure Only Unprivileged Read |
| 0b1111 | 0b000 | 0b0111 | 0b1000 | 0b111 | ATS12NSOUW | Address Translate Stages 1 and Non-secure Only Unprivileged Write |
| 0b1111 | 0b000 | 0b0111 | 0b1001 | 0b000 | ATS1CPRP | Address Translate Stage 1 Current stage PL1 Read PAN |
| 0b1111 | 0b000 | 0b0111 | 0b1001 | 0b001 | ATS1CPWP | Address Translate Stage 1 Current stage PL1 Write PAN |
| 0b1111 | 0b000 | 0b0111 | 0b1010 | 0b001 | DCCMVAC | Data Cache Line Clean by VA to PoC |
| 0b1111 | 0b000 | 0b0111 | 0b1010 | 0b010 | DCCSW | Data Cache Line Clean by Set/Way |
| 0b1111 | 0b000 | 0b0111 | 0b1010 | 0b100 | CP15DSB | Data Synchronization Barrier System instruction |
| 0b1111 | 0b000 | 0b0111 | 0b1010 | 0b101 | CP15DMB | Data Memory Barrier System instruction |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|-------|-----------------------------|---|
| | | CRn | CRm | | | |
| 0b1111 | 0b000 | 0b0111 | 0b1011 | 0b001 | DCCMVAU | Data Cache li Clean by VA t PoU |
| 0b1111 | 0b000 | 0b0111 | 0b1110 | 0b001 | DCCIMVAC | Data Cache li Clean and Invalidate by VA to PoC |
| 0b1111 | 0b000 | 0b0111 | 0b1110 | 0b010 | DCCISW | Data Cache li Clean and Invalidate by Set/Way |
| 0b1111 | 0b000 | 0b1000 | 0b0011 | 0b000 | TLBIALLIS | TLB Invalidat All, Inner Shareable |
| 0b1111 | 0b000 | 0b1000 | 0b0011 | 0b001 | TLBIMVAIS | TLB Invalidat by VA, Inner Shareable |
| 0b1111 | 0b000 | 0b1000 | 0b0011 | 0b010 | TLBIASIDIS | TLB Invalidat by ASID matc Inner Shareal |
| 0b1111 | 0b000 | 0b1000 | 0b0011 | 0b011 | TLBIMVAAIS | TLB Invalidat by VA, All ASI Inner Shareal |
| 0b1111 | 0b000 | 0b1000 | 0b0011 | 0b101 | TLBIMVALIS | TLB Invalidat by VA, Last level, Inner Shareable |
| 0b1111 | 0b000 | 0b1000 | 0b0011 | 0b111 | TLBIMVAALIS | TLB Invalidat by VA, All ASI Last level, Inner Shareal |
| 0b1111 | 0b000 | 0b1000 | 0b0101 | 0b000 | ITLBIALL | Instruction TL Invalidate All |
| 0b1111 | 0b000 | 0b1000 | 0b0101 | 0b001 | ITLBIMVA | Instruction TL Invalidate by VA |
| 0b1111 | 0b000 | 0b1000 | 0b0101 | 0b010 | ITLBIASID | Instruction TL Invalidate by ASID match |
| 0b1111 | 0b000 | 0b1000 | 0b0110 | 0b000 | DTLBIALL | Data TLB Invalidate All |
| 0b1111 | 0b000 | 0b1000 | 0b0110 | 0b001 | DTLBIMVA | Data TLB Invalidate by VA |
| 0b1111 | 0b000 | 0b1000 | 0b0110 | 0b010 | DTLBIASID | Data TLB Invalidate by ASID match |
| 0b1111 | 0b000 | 0b1000 | 0b0111 | 0b000 | TLBIALL | TLB Invalidat All |
| 0b1111 | 0b000 | 0b1000 | 0b0111 | 0b001 | TLBIMVA | TLB Invalidat by VA |
| 0b1111 | 0b000 | 0b1000 | 0b0111 | 0b010 | TLBIASID | TLB Invalidat by ASID matc |
| 0b1111 | 0b000 | 0b1000 | 0b0111 | 0b011 | TLBIMVAA | TLB Invalidat by VA, All ASI |
| 0b1111 | 0b000 | 0b1000 | 0b0111 | 0b101 | TLBIMVAL | TLB Invalidat by VA, Last level |
| 0b1111 | 0b000 | 0b1000 | 0b0111 | 0b111 | TLBIMVAAL | TLB Invalidat by VA, All ASI Last level |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|-------|----------------------------|---|
| | | CRn | CRm | | | |
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b000 | PMCR | Performance Monitors Control Register |
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b001 | PMCNTENSET | Performance Monitors Counter Enable Set register |
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b010 | PMCNTENCLR | Performance Monitors Counter Enable Clear register |
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b011 | PMOVSr | Performance Monitors Overflow Flag Status Register |
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b100 | PMSWINC | Performance Monitors Software Increment register |
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b101 | PMSELR | Performance Monitors Event Counter Selection Register |
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b110 | PMCEID0 | Performance Monitors Common Event Identification register 0 |
| 0b1111 | 0b000 | 0b1001 | 0b1100 | 0b111 | PMCEID1 | Performance Monitors Common Event Identification register 1 |
| 0b1111 | 0b000 | 0b1001 | 0b1101 | 0b000 | PMCCNTR | Performance Monitors Cycle Count Register |
| 0b1111 | 0b000 | 0b1001 | 0b1101 | 0b001 | PMXEVTYPER | Performance Monitors Selected Event Type Register |
| 0b1111 | 0b000 | 0b1001 | 0b1101 | 0b010 | PMXEVCNTR | Performance Monitors Selected Event Count Register |
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b000 | PMUSERENR | Performance Monitors User Enable Register |
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b001 | PMINTENSET | Performance Monitors Interrupt Enable Set register |
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b010 | PMINTENCLR | Performance Monitors Interrupt Enable Clear register |
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b011 | PMOVSSET | Performance Monitors Overflow Flag |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|------------|-----------------------------------|--|
| | | CRn | CRm | | | |
| | | | | | | Status Set register |
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b100 | PMCEID2 | Performance Monitors Common Event Identification register 2 |
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b101 | PMCEID3 | Performance Monitors Common Event Identification register 3 |
| 0b1111 | 0b000 | 0b1001 | 0b1110 | 0b110 | PMMIR | Performance Monitors Machine Identification Register |
| 0b1111 | 0b000 | 0b1010 | 0b0011 | 0b000 | AMAIRO | Auxiliary Memory Attribute Indirection Register 0 |
| 0b1111 | 0b000 | 0b1010 | 0b0011 | 0b001 | AMAIR1 | Auxiliary Memory Attribute Indirection Register 1 |
| 0b1111 | 0b000 | 0b1100 | 0b0000 | 0b000 | VBAR | Vector Base Address Register |
| 0b1111 | 0b000 | 0b1100 | 0b0000 | 0b010 | RMR | Reset Management Register |
| 0b1111 | 0b000 | 0b1100 | 0b0001 | 0b000 | ISR | Interrupt Status Register |
| 0b1111 | 0b000 | 0b1100 | 0b0001 | 0b001 | DISR | Deferred Interrupt Status Register |
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b000 | ICC_IAR0 | Interrupt Controller Interrupt Acknowledge Register 0 |
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b001 | ICC_EOIR0 | Interrupt Controller End Of Interrupt Register 0 |
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b010 | ICC_HPPIR0 | Interrupt Controller Highest Priority Pending Interrupt Register 0 |
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b011 | ICC_BPR0 | Interrupt Controller Binary Point Register 0 |
| 0b1111 | 0b000 | 0b1100 | 0b1000 | 0b1:n[1:0] | ICC_AP0R<n> | Interrupt Controller Active Priority Group 0 Registers |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|------------|-----------------------------------|--|
| | | CRn | CRm | | | |
| 0b1111 | 0b000 | 0b1100 | 0b1001 | 0b0:n[1:0] | ICC_AP1R<n> | Interrupt Controller Active Priority Group 1 Registers |
| 0b1111 | 0b000 | 0b1100 | 0b1011 | 0b001 | ICC_DIR | Interrupt Controller Deactivate Interrupt Register |
| 0b1111 | 0b000 | 0b1100 | 0b1011 | 0b011 | ICC_RPR | Interrupt Controller Running Priority Register |
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b000 | ICC_IAR1 | Interrupt Controller Interrupt Acknowledge Register 1 |
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b001 | ICC_EOIR1 | Interrupt Controller End Of Interrupt Register 1 |
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b010 | ICC_HPPIR1 | Interrupt Controller Highest Priority Pending Interrupt Register 1 |
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b011 | ICC_BPR1 | Interrupt Controller Binary Point Register 1 |
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b100 | ICC_CTLR | Interrupt Controller Control Register |
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b101 | ICC_SRE | Interrupt Controller System Register Enable register |
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b110 | ICC_IGRPEN0 | Interrupt Controller Interrupt Group 0 Enable register |
| 0b1111 | 0b000 | 0b1100 | 0b1100 | 0b111 | ICC_IGRPEN1 | Interrupt Controller Interrupt Group 1 Enable register |
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b000 | FCSEIDR | FCSE Process ID register |
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b001 | CONTEXTIDR | Context ID Register |
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b010 | TPIDRURW | PL0 Read/Write Software Thread ID Register |
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b011 | TPIDRURO | PL0 Read-Only Software |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|------------|--------|-------------------------------------|--|
| | | CRn | CRm | | | |
| | | | | | | Thread ID Register |
| 0b1111 | 0b000 | 0b1101 | 0b0000 | 0b100 | TPIDRPRW | PL1 Software Thread ID Register |
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b000 | AMCR | Activity Monitors Control Register |
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b001 | AMCFGR | Activity Monitors Configuration Register |
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b010 | AMCGCR | Activity Monitors Counter Group Configuration Register |
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b011 | AMUSERENR | Activity Monitors User Enable Register |
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b100 | AMCNTENCLR0 | Activity Monitors Counter Enable Clear Register 0 |
| 0b1111 | 0b000 | 0b1101 | 0b0010 | 0b101 | AMCNTENSET0 | Activity Monitors Counter Enable Set Register 0 |
| 0b1111 | 0b000 | 0b1101 | 0b0011 | 0b000 | AMCNTENCLR1 | Activity Monitors Counter Enable Clear Register 1 |
| 0b1111 | 0b000 | 0b1101 | 0b0011 | 0b001 | AMCNTENSET1 | Activity Monitors Counter Enable Set Register 1 |
| 0b1111 | 0b000 | 0b1101 | 0b011:n[3] | n[2:0] | AMEVTYPER0<n> | Activity Monitors Event Type Register 0 |
| 0b1111 | 0b000 | 0b1101 | 0b111:n[3] | n[2:0] | AMEVTYPER1<n> | Activity Monitors Event Type Register 1 |
| 0b1111 | 0b000 | 0b1110 | 0b0000 | 0b000 | CNTFRQ | Counter-timer Frequency register |
| 0b1111 | 0b000 | 0b1110 | 0b0001 | 0b000 | CNTKCTL | Counter-timer Kernel Control register |
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b000 | CNTP_TVAL | Counter-timer Physical Time TimerValue register |
| 0b1111 | 0b000 | 0b1110 | 0b0010 | 0b001 | CNTP_CTL | Counter-timer Physical Time Control register |
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b000 | CNTV_TVAL | Counter-timer Virtual Time TimerValue register |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|-------------|--------|--------------------------------------|--|
| | | CRn | CRm | | | |
| 0b1111 | 0b000 | 0b1110 | 0b0011 | 0b001 | CNTV_CTL | Counter-timer Virtual Timer Control register |
| 0b1111 | 0b000 | 0b1110 | 0b10:n[4:3] | n[2:0] | PMEVCNTR<n> | Performance Monitors Event Count Register |
| 0b1111 | 0b000 | 0b1110 | 0b1111 | 0b111 | PMCCFILTR | Performance Monitors Cycle Count Filter Register |
| 0b1111 | 0b000 | 0b1110 | 0b11:n[4:3] | n[2:0] | PMEVTYPEPER<n> | Performance Monitors Event Type Register |
| 0b1111 | 0b001 | 0b0000 | 0b0000 | 0b000 | CCSIDR | Current Cache Size ID Register |
| 0b1111 | 0b001 | 0b0000 | 0b0000 | 0b001 | CLIDR | Cache Level ID Register |
| 0b1111 | 0b001 | 0b0000 | 0b0000 | 0b010 | CCSIDR2 | Current Cache Size ID Register 2 |
| 0b1111 | 0b001 | 0b0000 | 0b0000 | 0b111 | AIDR | Auxiliary ID Register |
| 0b1111 | 0b010 | 0b0000 | 0b0000 | 0b000 | CSSELR | Cache Size Selection Register |
| 0b1111 | 0b011 | 0b0100 | 0b0101 | 0b000 | DSPSR | Debug Saved Program State Register |
| 0b1111 | 0b011 | 0b0100 | 0b0101 | 0b001 | DLR | Debug Link Register |
| 0b1111 | 0b100 | 0b0000 | 0b0000 | 0b000 | VPIDR | Virtualization Processor ID Register |
| 0b1111 | 0b100 | 0b0000 | 0b0000 | 0b101 | VMPIDR | Virtualization Multiprocessor ID Register |
| 0b1111 | 0b100 | 0b0001 | 0b0000 | 0b000 | HSCTLR | Hyp System Control Register |
| 0b1111 | 0b100 | 0b0001 | 0b0000 | 0b001 | HACTLR | Hyp Auxiliary Control Register |
| 0b1111 | 0b100 | 0b0001 | 0b0000 | 0b011 | HACTLR2 | Hyp Auxiliary Control Register 2 |
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b000 | HCR | Hyp Configuration Register |
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b001 | HDCR | Hyp Debug Control Register |
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b010 | HCPTR | Hyp Architectural Feature Trap Register |
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b011 | HSTR | Hyp System Trap Register |
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b100 | HCR2 | Hyp Configuration Register 2 |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|-------|------------------------------|---|
| | | CRn | CRm | | | |
| 0b1111 | 0b100 | 0b0001 | 0b0001 | 0b111 | HACR | Hyp Auxiliary Configuration Register |
| 0b1111 | 0b100 | 0b0001 | 0b0010 | 0b001 | HTRFCR | Hyp Trace Filter Control Register |
| 0b1111 | 0b100 | 0b0010 | 0b0000 | 0b010 | HTCR | Hyp Translation Control Register |
| 0b1111 | 0b100 | 0b0010 | 0b0001 | 0b010 | VTCR | Virtualization Translation Control Register |
| 0b1111 | 0b100 | 0b0101 | 0b0001 | 0b000 | HADEFSR | Hyp Auxiliary Data Fault Status Register |
| 0b1111 | 0b100 | 0b0101 | 0b0001 | 0b001 | HAIFSR | Hyp Auxiliary Instruction Fault Status Register |
| 0b1111 | 0b100 | 0b0101 | 0b0010 | 0b000 | HSR | Hyp Syndrome Register |
| 0b1111 | 0b100 | 0b0101 | 0b0010 | 0b011 | VDEFSR | Virtual SError Exception Syndrome Register |
| 0b1111 | 0b100 | 0b0110 | 0b0000 | 0b000 | HDFAR | Hyp Data Fault Address Register |
| 0b1111 | 0b100 | 0b0110 | 0b0000 | 0b010 | HIFAR | Hyp Instruction Fault Address Register |
| 0b1111 | 0b100 | 0b0110 | 0b0000 | 0b100 | HPFAR | Hyp IPA Fault Address Register |
| 0b1111 | 0b100 | 0b0111 | 0b1000 | 0b000 | ATS1HR | Address Translate Stage 1 Hyp mode Read |
| 0b1111 | 0b100 | 0b0111 | 0b1000 | 0b001 | ATS1HW | Address Translate Stage 1 Hyp mode Write |
| 0b1111 | 0b100 | 0b1000 | 0b0000 | 0b001 | TLBIIPAS2IS | TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable |
| 0b1111 | 0b100 | 0b1000 | 0b0000 | 0b101 | TLBIIPAS2LIS | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable |
| 0b1111 | 0b100 | 0b1000 | 0b0011 | 0b000 | TLBIALLHIS | TLB Invalidate All, Hyp mode Inner Shareable |
| 0b1111 | 0b100 | 0b1000 | 0b0011 | 0b001 | TLBIMVAHIS | TLB Invalidate by VA, Hyp mode, Inner Shareable |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|------------|-----------------------------------|--|
| | | CRn | CRm | | | |
| 0b1111 | 0b100 | 0b1000 | 0b0011 | 0b100 | TLBIALNSNHIS | TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable |
| 0b1111 | 0b100 | 0b1000 | 0b0011 | 0b101 | TLBIMVALHIS | TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable |
| 0b1111 | 0b100 | 0b1000 | 0b0100 | 0b001 | TLBIIPAS2 | TLB Invalidate by Intermediate Physical Address, Stage 2 |
| 0b1111 | 0b100 | 0b1000 | 0b0100 | 0b101 | TLBIIPAS2L | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level |
| 0b1111 | 0b100 | 0b1000 | 0b0111 | 0b000 | TLBIALLH | TLB Invalidate All, Hyp mode |
| 0b1111 | 0b100 | 0b1000 | 0b0111 | 0b001 | TLBIMVAH | TLB Invalidate by VA, Hyp mode |
| 0b1111 | 0b100 | 0b1000 | 0b0111 | 0b100 | TLBIALNSNH | TLB Invalidate All, Non-Secure Non-Hyp |
| 0b1111 | 0b100 | 0b1000 | 0b0111 | 0b101 | TLBIMVALH | TLB Invalidate by VA, Last level, Hyp mode |
| 0b1111 | 0b100 | 0b1010 | 0b0010 | 0b000 | HMAIR0 | Hyp Memory Attribute Indirection Register 0 |
| 0b1111 | 0b100 | 0b1010 | 0b0010 | 0b001 | HMAIR1 | Hyp Memory Attribute Indirection Register 1 |
| 0b1111 | 0b100 | 0b1010 | 0b0011 | 0b000 | HAMAIRO | Hyp Auxiliary Memory Attribute Indirection Register 0 |
| 0b1111 | 0b100 | 0b1010 | 0b0011 | 0b001 | HAMAIR1 | Hyp Auxiliary Memory Attribute Indirection Register 1 |
| 0b1111 | 0b100 | 0b1100 | 0b0000 | 0b000 | HVBAR | Hyp Vector Base Address Register |
| 0b1111 | 0b100 | 0b1100 | 0b0000 | 0b010 | HRMR | Hyp Reset Management Register |
| 0b1111 | 0b100 | 0b1100 | 0b0001 | 0b001 | VDISR | Virtual Deferred Interrupt Status Register |
| 0b1111 | 0b100 | 0b1100 | 0b1000 | 0b0:n[1:0] | ICH_AP0R<n> | Interrupt Controller Hyp Active Priority Group 0 Registers |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|------------|------------|-----------------------------------|---|
| | | CRn | CRm | | | |
| 0b1111 | 0b100 | 0b1100 | 0b1001 | 0b0:n[1:0] | ICH_AP1R<n> | Interrupt Controller Hypervisor Active Priority Group 1 Registers |
| 0b1111 | 0b100 | 0b1100 | 0b1001 | 0b101 | ICC_HSRE | Interrupt Controller Hypervisor System Register Enable register |
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b000 | ICH_HCR | Interrupt Controller Hypervisor Control Register |
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b001 | ICH_VTR | Interrupt Controller Virtual Machine Type Register |
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b010 | ICH_MISR | Interrupt Controller Maintenance Interrupt Status Register |
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b011 | ICH_EISR | Interrupt Controller Enable of Interrupt Status Register |
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b101 | ICH_ELSR | Interrupt Controller Empty List Register Status Register |
| 0b1111 | 0b100 | 0b1100 | 0b1011 | 0b111 | ICH_VMCR | Interrupt Controller Virtual Machine Control Register |
| 0b1111 | 0b100 | 0b1100 | 0b110:n[3] | n[2:0] | ICH_LR<n> | Interrupt Controller List Registers |
| 0b1111 | 0b100 | 0b1100 | 0b111:n[3] | n[2:0] | ICH_LRC<n> | Interrupt Controller List Registers |
| 0b1111 | 0b100 | 0b1101 | 0b0000 | 0b010 | HTPIDR | Hypervisor Software Thread ID Register |
| 0b1111 | 0b100 | 0b1110 | 0b0001 | 0b000 | CNTHCTL | Counter-timer Hypervisor Control register |
| 0b1111 | 0b100 | 0b1110 | 0b0010 | 0b000 | CNTHP_TVAL | Counter-timer Hypervisor Physical Timer TimerValue register |
| 0b1111 | 0b100 | 0b1110 | 0b0010 | 0b001 | CNTHP_CTL | Counter-timer Hypervisor Physical Timer Control register |
| 0b1111 | 0b110 | 0b1100 | 0b1100 | 0b100 | ICC_MCTLR | Interrupt Controller Monitor Control Register |
| 0b1111 | 0b110 | 0b1100 | 0b1100 | 0b101 | ICC_MSRE | Interrupt Controller |

| coproc | opc1 | Register selectors | | opc2 | Name | Description |
|--------|-------|--------------------|--------|-------|-----------------------------|--|
| | | CRn | CRm | | | |
| | | | | | | Monitor System Register Enable register |
| 0b1111 | 0b110 | 0b1100 | 0b1100 | 0b111 | ICC_MGRPEN1 | Interrupt Controller Monitor Interrupt Group 1 Enable register |

Accessed using MCRR/MRRC:

| coproc | Register selectors | | Name | Description |
|--------|--------------------|------------|------------------------------------|--|
| | CRm | opc1 | | |
| 0b1110 | 0b0001 | 0b0000 | DBGDRAR | Debug ROM Address Register |
| 0b1110 | 0b0010 | 0b0000 | DBGDSAR | Debug Self Address Register |
| 0b1111 | 0b000:n[3] | 0b0:n[2:0] | AMEVCNTR0<n> | Activity Monitors Event Counter Registers 0 |
| 0b1111 | 0b0010 | 0b0000 | TTBR0 | Translation Table Base Register 0 |
| 0b1111 | 0b0010 | 0b0001 | TTBR1 | Translation Table Base Register 1 |
| 0b1111 | 0b0010 | 0b0100 | HTTBR | Hyp Translation Table Base Register |
| 0b1111 | 0b0010 | 0b0110 | VTTBR | Virtualization Translation Table Base Register |
| 0b1111 | 0b010:n[3] | 0b0:n[2:0] | AMEVCNTR1<n> | Activity Monitors Event Counter Registers 1 |
| 0b1111 | 0b0111 | 0b0000 | PAR | Physical Address Register |
| 0b1111 | 0b1001 | 0b0000 | PMCCNTR | Performance Monitors Cycle Count Register |
| 0b1111 | 0b1100 | 0b0000 | ICC_SGI1R | Interrupt Controller Software Generated Interrupt Group 1 Register |
| 0b1111 | 0b1100 | 0b0001 | ICC_ASGI1R | Interrupt Controller Alias Software Generated Interrupt Group 1 Register |
| 0b1111 | 0b1100 | 0b0010 | ICC_SGI0R | Interrupt Controller Software Generated Interrupt Group 0 Register |
| 0b1111 | 0b1110 | 0b0000 | CNTPCT | Counter-timer Physical Count register |
| 0b1111 | 0b1110 | 0b0001 | CNTVCT | Counter-timer Virtual Count register |
| 0b1111 | 0b1110 | 0b0010 | CNTP_CVAL | Counter-timer Physical Timer CompareValue register |
| 0b1111 | 0b1110 | 0b0011 | CNTV_CVAL | Counter-timer Virtual Timer CompareValue register |
| 0b1111 | 0b1110 | 0b0100 | CNTVOFF | Counter-timer Virtual Offset register |
| 0b1111 | 0b1110 | 0b0110 | CNTHP_CVAL | Counter-timer Hyp Physical CompareValue register |
| 0b1111 | 0b1110 | 0b1000 | CNTPCTSS | Counter-timer Self-Synchronized Physical Count register |
| 0b1111 | 0b1110 | 0b1001 | CNTVCTSS | Counter-timer Self-Synchronized Virtual Count register |

Accessed using MRS/MSR:

| Register selectors | | | Name | Description |
|--------------------|-----|--------|-------------------------|------------------------------------|
| R | M | M1 | | |
| 0b0 | 0b1 | 0b1110 | ELR_hyp | Exception Link Register (Hyp mode) |

| Register selectors | | | Name | Description |
|--------------------|-----|--------|--------------------------|---|
| R | M | M1 | | |
| 0b1 | 0b0 | 0b1110 | SPSR_fiq | Saved Program Status Register (FIQ mode) |
| 0b1 | 0b1 | 0b0000 | SPSR_irq | Saved Program Status Register (IRQ mode) |
| 0b1 | 0b1 | 0b0010 | SPSR_svc | Saved Program Status Register (Supervisor mode) |
| 0b1 | 0b1 | 0b0100 | SPSR_abt | Saved Program Status Register (Abort mode) |
| 0b1 | 0b1 | 0b0110 | SPSR_und | Saved Program Status Register (Undefined mode) |
| 0b1 | 0b1 | 0b1100 | SPSR_mon | Saved Program Status Register (Monitor mode) |
| 0b1 | 0b1 | 0b1110 | SPSR_hyp | Saved Program Status Register (Hyp mode) |

Accessed using VMRS/VMSR:

| Register selectors reg | Name | Description |
|---------------------------|-----------------------|--|
| 0b0000 | FPSID | Floating-Point System ID register |
| 0b0001 | FPSCR | Floating-Point Status and Control Register |
| 0b0101 | MVFR2 | Media and VFP Feature Register 2 |
| 0b0110 | MVFR1 | Media and VFP Feature Register 1 |
| 0b0111 | MVFR0 | Media and VFP Feature Register 0 |
| 0b1000 | FPEXC | Floating-Point Exception Control register |

Registers and operations in AArch64

Accessed using AT:

| Register selectors | | | | Name | Description |
|--------------------|-------|--------|--------|---------------------------|--|
| op0 | op1 | CRn | CRm | | |
| 0b01 | 0b000 | 0b0111 | 0b1000 | AT S1E1R | Address Translate Stage 1 EL1 Read |
| 0b01 | 0b000 | 0b0111 | 0b1000 | AT S1E1W | Address Translate Stage 1 EL1 Write |
| 0b01 | 0b000 | 0b0111 | 0b1000 | AT S1E0R | Address Translate Stage 1 EL0 Read |
| 0b01 | 0b000 | 0b0111 | 0b1000 | AT S1E0W | Address Translate Stage 1 EL0 Write |
| 0b01 | 0b000 | 0b0111 | 0b1001 | AT S1E1RP | Address Translate Stage 1 EL1 Read PAN |
| 0b01 | 0b000 | 0b0111 | 0b1001 | AT S1E1WP | Address Translate Stage 1 EL1 Write PAN |
| 0b01 | 0b100 | 0b0111 | 0b1000 | AT S1E2R | Address Translate Stage 1 EL2 Read |
| 0b01 | 0b100 | 0b0111 | 0b1000 | AT S1E2W | Address Translate Stage 1 EL2 Write |
| 0b01 | 0b100 | 0b0111 | 0b1000 | AT S12E1R | Address Translate Stages 1 and 2 EL1 Read |
| 0b01 | 0b100 | 0b0111 | 0b1000 | AT S12E1W | Address Translate Stages 1 and 2 EL1 Write |
| 0b01 | 0b100 | 0b0111 | 0b1000 | AT S12E0R | Address Translate Stages 1 and 2 EL0 Read |
| 0b01 | 0b100 | 0b0111 | 0b1000 | AT S12E0W | Address Translate Stages 1 and 2 EL0 Write |
| 0b01 | 0b110 | 0b0111 | 0b1000 | AT S1E3R | Address Translate Stage 1 EL3 Read |
| 0b01 | 0b110 | 0b0111 | 0b1000 | AT S1E3W | Address Translate Stage 1 EL3 Write |

Accessed using BRB:

| Register selectors | | | | op2 | Name | Description |
|--------------------|-------|--------|--------|-------|--------------------------|---|
| op0 | op1 | CRn | CRm | | | |
| 0b01 | 0b001 | 0b0111 | 0b0010 | 0b100 | BRB IALL | Invalidate the Branch Record Buffer |
| 0b01 | 0b001 | 0b0111 | 0b0010 | 0b101 | BRB INJ | Branch Record Injection into the Branch Record Buffer |

Accessed using CFP:

| Register selectors | | | | op2 | Name | Description |
|--------------------|-------|--------|--------|-------|--------------------------|--|
| op0 | op1 | CRn | CRm | | | |
| 0b01 | 0b011 | 0b0111 | 0b0011 | 0b100 | CFP RCTX | Control Flow Prediction Restriction by Context |

Accessed using CPP:

| Register selectors | | | | op2 | Name | Description |
|--------------------|-------|--------|--------|-------|--------------------------|--|
| op0 | op1 | CRn | CRm | | | |
| 0b01 | 0b011 | 0b0111 | 0b0011 | 0b111 | CPP RCTX | Cache Prefetch Prediction Restriction by Context |

Accessed using DC:

| Register selectors | | | | op2 | Name | Description |
|--------------------|-------|--------|--------|-------|---------------------------|---|
| op0 | op1 | CRn | CRm | | | |
| 0b01 | 0b000 | 0b0111 | 0b0110 | 0b001 | DC IVAC | Data or unified Cache line Invalidate by VA to PoC |
| 0b01 | 0b000 | 0b0111 | 0b0110 | 0b010 | DC ISW | Data or unified Cache line Invalidate by Set/Way |
| 0b01 | 0b000 | 0b0111 | 0b0110 | 0b011 | DC IGVAC | Invalidate of Allocation Tags by VA to PoC |
| 0b01 | 0b000 | 0b0111 | 0b0110 | 0b100 | DC IGSW | Invalidate of Allocation Tags by Set/Way |
| 0b01 | 0b000 | 0b0111 | 0b0110 | 0b101 | DC IGDVAC | Invalidate of Data and Allocation Tags by VA to PoC |
| 0b01 | 0b000 | 0b0111 | 0b0110 | 0b110 | DC IGDSW | Invalidate of Data and Allocation Tags by Set/Way |
| 0b01 | 0b000 | 0b0111 | 0b1010 | 0b010 | DC CSW | Data or unified Cache line Clean by Set/Way |
| 0b01 | 0b000 | 0b0111 | 0b1010 | 0b100 | DC CGSW | Clean of Allocation Tags by Set/Way |
| 0b01 | 0b000 | 0b0111 | 0b1010 | 0b110 | DC CGDSW | Clean of Data and Allocation Tags by Set/Way |
| 0b01 | 0b000 | 0b0111 | 0b1110 | 0b010 | DC CISW | Data or unified Cache line Clean and Invalidate by Set/Way |
| 0b01 | 0b000 | 0b0111 | 0b1110 | 0b100 | DC CIGSW | Clean and Invalidate of Allocation Tags by Set/Way |
| 0b01 | 0b000 | 0b0111 | 0b1110 | 0b110 | DC CIGDSW | Clean and Invalidate of Data and Allocation Tags by Set/Way |
| 0b01 | 0b011 | 0b0111 | 0b0100 | 0b001 | DC ZVA | Data Cache Zero by VA |
| 0b01 | 0b011 | 0b0111 | 0b0100 | 0b011 | DC GVA | Data Cache set Allocation Tag by VA |
| 0b01 | 0b011 | 0b0111 | 0b0100 | 0b100 | DC GZVA | Data Cache set Allocation Tags and Zero by VA |
| 0b01 | 0b011 | 0b0111 | 0b1010 | 0b001 | DC CVAC | Data or unified Cache line Clean by VA to PoC |

| Register selectors | | | | op2 | Name | Description |
|--------------------|-------|--------|--------|-------|-----------------------------|--|
| op0 | op1 | CRn | CRm | | | |
| 0b01 | 0b011 | 0b0111 | 0b1010 | 0b011 | DC CGVAC | Clean of Allocation Tags by VA to PoC |
| 0b01 | 0b011 | 0b0111 | 0b1010 | 0b101 | DC CGDVAC | Clean of Data and Allocation Tags by VA to PoC |
| 0b01 | 0b011 | 0b0111 | 0b1011 | 0b001 | DC CVAU | Data or unified Cache line Clean by VA to PoU |
| 0b01 | 0b011 | 0b0111 | 0b1100 | 0b001 | DC CVAP | Data or unified Cache line Clean by VA to PoP |
| 0b01 | 0b011 | 0b0111 | 0b1100 | 0b011 | DC CGVAP | Clean of Allocation Tags by VA to PoP |
| 0b01 | 0b011 | 0b0111 | 0b1100 | 0b101 | DC CGDVAP | Clean of Data and Allocation Tags by VA to PoP |
| 0b01 | 0b011 | 0b0111 | 0b1101 | 0b001 | DC CVADP | Data or unified Cache line Clean by VA to PoDP |
| 0b01 | 0b011 | 0b0111 | 0b1101 | 0b011 | DC CGVADP | Clean of Allocation Tags by VA to PoDP |
| 0b01 | 0b011 | 0b0111 | 0b1101 | 0b101 | DC CGDVADP | Clean of Data and Allocation Tags by VA to PoDP |
| 0b01 | 0b011 | 0b0111 | 0b1110 | 0b001 | DC CIVAC | Data or unified Cache line Clean and Invalidate by VA to PoC |
| 0b01 | 0b011 | 0b0111 | 0b1110 | 0b011 | DC CIGVAC | Clean and Invalidate of Allocation Tags by VA to PoC |
| 0b01 | 0b011 | 0b0111 | 0b1110 | 0b101 | DC CIGDVAC | Clean and Invalidate of Data and Allocation Tags by VA to PoC |
| 0b01 | 0b110 | 0b0111 | 0b1110 | 0b001 | DC CIPAPA | Data or unified Cache line Clean and Invalidate by PA to PoPA |
| 0b01 | 0b110 | 0b0111 | 0b1110 | 0b101 | DC CIGDPAPA | Clean and Invalidate of Data and Allocation Tags by PA to PoPA |

Accessed using DVP:

| Register selectors | | | | op2 | Name | Description |
|--------------------|-------|--------|--------|-------|--------------------------|--|
| op0 | op1 | CRn | CRm | | | |
| 0b01 | 0b011 | 0b0111 | 0b0011 | 0b101 | DVP RCTX | Data Value Prediction Restriction by Context |

Accessed using IC:

| Register selectors | | | | op2 | Name | Description |
|--------------------|-------|--------|--------|-------|----------------------------|--|
| op0 | op1 | CRn | CRm | | | |
| 0b01 | 0b000 | 0b0111 | 0b0001 | 0b000 | IC IALLUIS | Instruction Cache Invalidate All to PoU, Inner Shareable |
| 0b01 | 0b000 | 0b0111 | 0b0101 | 0b000 | IC IALLU | Instruction Cache Invalidate All to PoU |
| 0b01 | 0b011 | 0b0111 | 0b0101 | 0b001 | IC IVAU | Instruction Cache line Invalidate by VA to PoU |

Accessed using MRS/MSR:

| Register selectors | | | | op2 | Name | Description |
|--------------------|-------|--------|--------|-------|-----------------------------|-----------------------------------|
| op0 | op1 | CRn | CRm | | | |
| 0b10 | 0b000 | 0b0000 | 0b0000 | 0b010 | OSDTRRX_EL1 | OS Lock Transfer Register Receive |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|-------------------------------------|--|
| | | CRn | CRm | | | |
| 0b10 | 0b000 | 0b0000 | 0b0010 | 0b000 | MDCCINT_EL1 | Monitor Interrupt Enable Register |
| 0b10 | 0b000 | 0b0000 | 0b0010 | 0b010 | MDSCR_EL1 | Monitor System Control Register |
| 0b10 | 0b000 | 0b0000 | 0b0011 | 0b010 | OSDTRTX_EL1 | OS Lock Transfer Register Transmitter |
| 0b10 | 0b000 | 0b0000 | 0b0110 | 0b010 | OSECCR_EL1 | OS Lock Exception Catch Control Register |
| 0b10 | 0b000 | 0b0000 | n[3:0] | 0b100 | DBGBVR<n>_EL1 | Debug Breakpoint Value Register |
| 0b10 | 0b000 | 0b0000 | n[3:0] | 0b101 | DBGBCR<n>_EL1 | Debug Breakpoint Control Register |
| 0b10 | 0b000 | 0b0000 | n[3:0] | 0b110 | DBGWVR<n>_EL1 | Debug Watchpoint Value Register |
| 0b10 | 0b000 | 0b0000 | n[3:0] | 0b111 | DBGWCR<n>_EL1 | Debug Watchpoint Control Register |
| 0b10 | 0b000 | 0b0001 | 0b0000 | 0b000 | MDRAR_EL1 | Monitor ROM Address Register |
| 0b10 | 0b000 | 0b0001 | 0b0000 | 0b100 | OSLAR_EL1 | OS Lock Register |
| 0b10 | 0b000 | 0b0001 | 0b0001 | 0b100 | OSLSR_EL1 | OS Lock Register |
| 0b10 | 0b000 | 0b0001 | 0b0011 | 0b100 | OSDLR_EL1 | OS Double Lock Register |
| 0b10 | 0b000 | 0b0001 | 0b0100 | 0b100 | DBGPRCR_EL1 | Debug Point Control Register |
| 0b10 | 0b000 | 0b0111 | 0b1000 | 0b110 | DBGCLAIMSET_EL1 | Debug Claim Tag Set Register |
| 0b10 | 0b000 | 0b0111 | 0b1001 | 0b110 | DBGCLAIMCLR_EL1 | Debug Claim Tag Clear Register |
| 0b10 | 0b000 | 0b0111 | 0b1110 | 0b110 | DBGAUTHSTATUS_EL1 | Debug Authentication Status Register |
| 0b10 | 0b001 | 0b0000 | 0b0000 | 0b001 | TRCTRACEIDR | Trace ID Register |
| 0b10 | 0b001 | 0b0000 | 0b0000 | 0b010 | TRCVICTLR | View Instruction Control Register |
| 0b10 | 0b001 | 0b0000 | 0b0000 | 0b110 | TRCIDR8 | ID Register |
| 0b10 | 0b001 | 0b0000 | 0b0000 | 0b111 | TRCIMSPEC0 | IMP Debug Register |
| 0b10 | 0b001 | 0b0000 | 0b0001 | 0b000 | TRCPRGCTLR | Program Control Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|-------------|-------|--------------------------------------|--|
| | | CRn | CRm | | | |
| 0b10 | 0b001 | 0b0000 | 0b0001 | 0b001 | TRCQCTLR | Q Element Control Register |
| 0b10 | 0b001 | 0b0000 | 0b0001 | 0b010 | TRCVIIECTLR | ViewInst Include/Control Register |
| 0b10 | 0b001 | 0b0000 | 0b0001 | 0b110 | TRCIDR9 | ID Register |
| 0b10 | 0b001 | 0b0000 | 0b0010 | 0b010 | TRCVISSCTLR | ViewInst Stop Control Register |
| 0b10 | 0b001 | 0b0000 | 0b0010 | 0b110 | TRCIDR10 | ID Register |
| 0b10 | 0b001 | 0b0000 | 0b0011 | 0b000 | TRCSTATR | Trace Status Register |
| 0b10 | 0b001 | 0b0000 | 0b0011 | 0b010 | TRCVIPCSSCTLR | ViewInst Stop PE Comparison Control Register |
| 0b10 | 0b001 | 0b0000 | 0b0011 | 0b110 | TRCIDR11 | ID Register |
| 0b10 | 0b001 | 0b0000 | 0b00:n[1:0] | 0b100 | TRCSEQEVR<n> | Sequencer State Trace Control Register |
| 0b10 | 0b001 | 0b0000 | 0b00:n[1:0] | 0b101 | TRCCNTRLDVR<n> | Counter Value Register <n> |
| 0b10 | 0b001 | 0b0000 | 0b0100 | 0b000 | TRCCONFIGR | Trace Configuration Register |
| 0b10 | 0b001 | 0b0000 | 0b0100 | 0b110 | TRCIDR12 | ID Register |
| 0b10 | 0b001 | 0b0000 | 0b0101 | 0b110 | TRCIDR13 | ID Register |
| 0b10 | 0b001 | 0b0000 | 0b0110 | 0b000 | TRCAUXCTLR | Auxiliary Control Register |
| 0b10 | 0b001 | 0b0000 | 0b0110 | 0b100 | TRCSEQRSTEVR | Sequencer Reset Control Register |
| 0b10 | 0b001 | 0b0000 | 0b0111 | 0b100 | TRCSEQSTR | Sequencer State Register |
| 0b10 | 0b001 | 0b0000 | 0b01:n[1:0] | 0b101 | TRCCNTCTLR<n> | Counter Control Register |
| 0b10 | 0b001 | 0b0000 | 0b0:n[2:0] | 0b111 | TRCIMSPEC<n> | IMP DEL Register |
| 0b10 | 0b001 | 0b0000 | 0b1000 | 0b000 | TRCEVENTCTL0R | Event Control Register |
| 0b10 | 0b001 | 0b0000 | 0b1000 | 0b111 | TRCIDR0 | ID Register |
| 0b10 | 0b001 | 0b0000 | 0b1001 | 0b000 | TRCEVENTCTL1R | Event Control Register |
| 0b10 | 0b001 | 0b0000 | 0b1001 | 0b111 | TRCIDR1 | ID Register |
| 0b10 | 0b001 | 0b0000 | 0b1010 | 0b000 | TRCRSR | Resource Status Register |
| 0b10 | 0b001 | 0b0000 | 0b1010 | 0b111 | TRCIDR2 | ID Register |
| 0b10 | 0b001 | 0b0000 | 0b1011 | 0b000 | TRCSTALLCTLR | Stall Control Register |
| 0b10 | 0b001 | 0b0000 | 0b1011 | 0b111 | TRCIDR3 | ID Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|-------------|-----------|--------------------------------------|--|
| | | CRn | CRm | | | |
| 0b10 | 0b001 | 0b0000 | 0b10:n[1:0] | 0b100 | TRCEXTINSEL<n> | External Select Register <n> |
| 0b10 | 0b001 | 0b0000 | 0b10:n[1:0] | 0b101 | TRCCNTVR<n> | Counter Register <n> |
| 0b10 | 0b001 | 0b0000 | 0b1100 | 0b000 | TRCTSCTLR | Timestamp Control Register |
| 0b10 | 0b001 | 0b0000 | 0b1100 | 0b111 | TRCIDR4 | ID Register 4 |
| 0b10 | 0b001 | 0b0000 | 0b1101 | 0b000 | TRCSYNCPR | Synchronous Period Register |
| 0b10 | 0b001 | 0b0000 | 0b1101 | 0b111 | TRCIDR5 | ID Register 5 |
| 0b10 | 0b001 | 0b0000 | 0b1110 | 0b000 | TRCCCCTLR | Cycle Control Register |
| 0b10 | 0b001 | 0b0000 | 0b1110 | 0b111 | TRCIDR6 | ID Register 6 |
| 0b10 | 0b001 | 0b0000 | 0b1111 | 0b000 | TRCBBCTLR | Branch Broadcast Control Register |
| 0b10 | 0b001 | 0b0000 | 0b1111 | 0b111 | TRCIDR7 | ID Register 7 |
| 0b10 | 0b001 | 0b0001 | 0b0001 | 0b100 | TRCOSLSR | Trace OS Status Register |
| 0b10 | 0b001 | 0b0001 | 0b0:n[2:0] | 0b010 | TRCSSCCR<n> | Single-shot Comparator Control Register <n> |
| 0b10 | 0b001 | 0b0001 | 0b0:n[2:0] | 0b011 | TRCSSPCICR<n> | Single-shot Processing Element Comparator Input Control Register <n> |
| 0b10 | 0b001 | 0b0001 | 0b1:n[2:0] | 0b010 | TRCSSCSR<n> | Single-shot Comparator Control Register <n> |
| 0b10 | 0b001 | 0b0001 | n[3:0] | 0b00:n[4] | TRCRSCTLR<n> | Resource Selection Control Register <n> |
| 0b10 | 0b001 | 0b0010 | n[2:0]:0b0 | 0b00:n[3] | TRCACVR<n> | Address Comparator Value Register <n> |
| 0b10 | 0b001 | 0b0010 | n[2:0]:0b0 | 0b01:n[3] | TRCACATR<n> | Address Comparator Access Threshold Register <n> |
| 0b10 | 0b001 | 0b0011 | 0b0000 | 0b010 | TRCCIDCCTLR0 | Context Identifier Comparator Control Register 0 |
| 0b10 | 0b001 | 0b0011 | 0b0001 | 0b010 | TRCCIDCCTLR1 | Context Identifier Comparator Control Register 1 |
| 0b10 | 0b001 | 0b0011 | 0b0010 | 0b010 | TRCVMIDCCTLR0 | Virtual Context Identifier Control Register 0 |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|------------|-----------|-------------------------------------|---|
| | | CRn | CRm | | | |
| | | | | | | Comparison Control Register |
| 0b10 | 0b001 | 0b0011 | 0b0011 | 0b010 | TRCVMIDCCTLR1 | Virtual C Identifier Comparison Control Register |
| 0b10 | 0b001 | 0b0011 | n[2:0]:0b0 | 0b000 | TRCCIDCVR<n> | Context Identifier Comparison Value Register <n> |
| 0b10 | 0b001 | 0b0011 | n[2:0]:0b0 | 0b001 | TRCVMIDCVR<n> | Virtual C Identifier Comparison Value Register <n> |
| 0b10 | 0b001 | 0b0111 | 0b0010 | 0b111 | TRCDEVID | Device Configuration Register |
| 0b10 | 0b001 | 0b0111 | 0b1000 | 0b110 | TRCCLAIMSET | Claim Tag Register |
| 0b10 | 0b001 | 0b0111 | 0b1001 | 0b110 | TRCCLAIMCLR | Claim Tag Register |
| 0b10 | 0b001 | 0b0111 | 0b1110 | 0b110 | TRCAUTHSTATUS | Authentication Status Register |
| 0b10 | 0b001 | 0b0111 | 0b1111 | 0b110 | TRCDEVARCH | Device Architecture Register |
| 0b10 | 0b001 | 0b1000 | n[3:0] | n[4]:0b00 | BRBINF<n>_EL1 | Branch Instruction Buffer Information Register |
| 0b10 | 0b001 | 0b1000 | n[3:0] | n[4]:0b01 | BRBSRC<n>_EL1 | Branch Instruction Buffer Source Address Register |
| 0b10 | 0b001 | 0b1000 | n[3:0] | n[4]:0b10 | BRBTGT<n>_EL1 | Branch Instruction Buffer Target Address Register |
| 0b10 | 0b001 | 0b1001 | 0b0000 | 0b000 | BRBCR_EL1 | Branch Instruction Buffer Control Register |
| 0b10 | 0b001 | 0b1001 | 0b0000 | 0b001 | BRBFCR_EL1 | Branch Instruction Buffer Filter Control Register |
| 0b10 | 0b001 | 0b1001 | 0b0000 | 0b010 | BRBTS_EL1 | Branch Instruction Buffer Timestamp Register |
| 0b10 | 0b001 | 0b1001 | 0b0001 | 0b000 | BRBINFINJ_EL1 | Branch Instruction Buffer Information Injection Register |
| 0b10 | 0b001 | 0b1001 | 0b0001 | 0b001 | BRBSRCINJ_EL1 | Branch Instruction Buffer Source Address Injection Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|-------------------------------|--|
| | | CRn | CRm | | | |
| 0b10 | 0b001 | 0b1001 | 0b0001 | 0b010 | BRBTGTINJ_EL1 | Branch Instruction Buffer Tag Address Injection Register |
| 0b10 | 0b001 | 0b1001 | 0b0010 | 0b000 | BRBIDR0_EL1 | Branch Instruction Buffer ID Register |
| 0b10 | 0b011 | 0b0000 | 0b0001 | 0b000 | MDCCSR_EL0 | Monitor Status Register |
| 0b10 | 0b011 | 0b0000 | 0b0100 | 0b000 | DBGDTR_EL0 | Debug Data Transfer Register duplex |
| 0b10 | 0b011 | 0b0000 | 0b0101 | 0b000 | DBGDTRRX_EL0 | Debug Data Transfer Register Receive |
| 0b10 | 0b011 | 0b0000 | 0b0101 | 0b000 | DBGDTRTX_EL0 | Debug Data Transfer Register Transmitter |
| 0b10 | 0b100 | 0b0000 | 0b0111 | 0b000 | DBGVCR32_EL2 | Debug Viewpoint Catch Register |
| 0b10 | 0b100 | 0b1001 | 0b0000 | 0b000 | BRBCR_EL2 | Branch Instruction Buffer Control Register |
| 0b11 | 0b000 | 0b0000 | 0b0000 | 0b000 | MIDR_EL1 | Main ID Register |
| 0b11 | 0b000 | 0b0000 | 0b0000 | 0b101 | MPIDR_EL1 | Multiprocessor Affinity Register |
| 0b11 | 0b000 | 0b0000 | 0b0000 | 0b110 | REVIDR_EL1 | Revision ID Register |
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b000 | ID_PFR0_EL1 | AArch32 Processor Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b001 | ID_PFR1_EL1 | AArch32 Processor Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b010 | ID_DFR0_EL1 | AArch32 Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b011 | ID_AFR0_EL1 | AArch32 Auxiliary Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b100 | ID_MMFR0_EL1 | AArch32 Memory Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b101 | ID_MMFR1_EL1 | AArch32 Memory Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b110 | ID_MMFR2_EL1 | AArch32 Memory Feature Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|---------------------------------|--|
| | | CRn | CRm | | | |
| 0b11 | 0b000 | 0b0000 | 0b0001 | 0b111 | ID_MMFR3_EL1 | AArch32 Memory Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b000 | ID_ISAR0_EL1 | AArch32 Instruction Attribute Register |
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b001 | ID_ISAR1_EL1 | AArch32 Instruction Attribute Register |
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b010 | ID_ISAR2_EL1 | AArch32 Instruction Attribute Register |
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b011 | ID_ISAR3_EL1 | AArch32 Instruction Attribute Register |
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b100 | ID_ISAR4_EL1 | AArch32 Instruction Attribute Register |
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b101 | ID_ISAR5_EL1 | AArch32 Instruction Attribute Register |
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b110 | ID_MMFR4_EL1 | AArch32 Memory Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0010 | 0b111 | ID_ISAR6_EL1 | AArch32 Instruction Attribute Register |
| 0b11 | 0b000 | 0b0000 | 0b0011 | 0b000 | MVFR0_EL1 | AArch32 and VFP Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0011 | 0b001 | MVFR1_EL1 | AArch32 and VFP Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0011 | 0b010 | MVFR2_EL1 | AArch32 and VFP Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0011 | 0b100 | ID_PFR2_EL1 | AArch32 Processor Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0011 | 0b101 | ID_DFR1_EL1 | Debug Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0011 | 0b110 | ID_MMFR5_EL1 | AArch32 Memory Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0100 | 0b000 | ID_AA64PFR0_EL1 | AArch64 Processor Feature Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|----------------------------------|--|
| | | CRn | CRm | | | |
| 0b11 | 0b000 | 0b0000 | 0b0100 | 0b001 | ID_AA64PFR1_EL1 | AArch64 Process Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0100 | 0b100 | ID_AA64ZFR0_EL1 | SVE Feature register |
| 0b11 | 0b000 | 0b0000 | 0b0100 | 0b101 | ID_AA64SMFR0_EL1 | SME Feature register |
| 0b11 | 0b000 | 0b0000 | 0b0101 | 0b000 | ID_AA64DFR0_EL1 | AArch64 Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0101 | 0b001 | ID_AA64DFR1_EL1 | AArch64 Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0101 | 0b100 | ID_AA64AFR0_EL1 | AArch64 Auxiliary Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0101 | 0b101 | ID_AA64AFR1_EL1 | AArch64 Auxiliary Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0110 | 0b000 | ID_AA64ISAR0_EL1 | AArch64 Instruction Attribute Register |
| 0b11 | 0b000 | 0b0000 | 0b0110 | 0b001 | ID_AA64ISAR1_EL1 | AArch64 Instruction Attribute Register |
| 0b11 | 0b000 | 0b0000 | 0b0110 | 0b010 | ID_AA64ISAR2_EL1 | AArch64 Instruction Attribute Register |
| 0b11 | 0b000 | 0b0000 | 0b0111 | 0b000 | ID_AA64MMFR0_EL1 | AArch64 Memory Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0111 | 0b001 | ID_AA64MMFR1_EL1 | AArch64 Memory Feature Register |
| 0b11 | 0b000 | 0b0000 | 0b0111 | 0b010 | ID_AA64MMFR2_EL1 | AArch64 Memory Feature Register |
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b000 | SCTLR_EL1 | System Control Register |
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b001 | ACTLR_EL1 | Auxiliary Control Register |
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b010 | CPACR_EL1 | Architectural Feature Control Register |
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b101 | RGSRR_EL1 | Random Allocation Seed Register |
| 0b11 | 0b000 | 0b0001 | 0b0000 | 0b110 | GCR_EL1 | Tag Control Register |
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b000 | ZCR_EL1 | SVE Control Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|-------------------------------|--|
| | | CRn | CRm | | | |
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b001 | TRFCR_EL1 | Trace Filter Control Register |
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b100 | SMPRI_EL1 | Streaming Mode Priority Register |
| 0b11 | 0b000 | 0b0001 | 0b0010 | 0b110 | SMCR_EL1 | SME Control Register |
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b000 | TTBR0_EL1 | Translation Table Base Register (EL1) |
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b001 | TTBR1_EL1 | Translation Table Base Register (EL1) |
| 0b11 | 0b000 | 0b0010 | 0b0000 | 0b010 | TCR_EL1 | Translation Control Register |
| 0b11 | 0b000 | 0b0010 | 0b0001 | 0b000 | APIAKeyLo_EL1 | Pointer Authentication Key A for Instructions (bits[63:0]) |
| 0b11 | 0b000 | 0b0010 | 0b0001 | 0b001 | APIAKeyHi_EL1 | Pointer Authentication Key A for Instructions (bits[127:64]) |
| 0b11 | 0b000 | 0b0010 | 0b0001 | 0b010 | APIBKeyLo_EL1 | Pointer Authentication Key B for Instructions (bits[63:0]) |
| 0b11 | 0b000 | 0b0010 | 0b0001 | 0b011 | APIBKeyHi_EL1 | Pointer Authentication Key B for Instructions (bits[127:64]) |
| 0b11 | 0b000 | 0b0010 | 0b0010 | 0b000 | APDAKeyLo_EL1 | Pointer Authentication Key A for Data Accesses (bits[63:0]) |
| 0b11 | 0b000 | 0b0010 | 0b0010 | 0b001 | APDAKeyHi_EL1 | Pointer Authentication Key A for Data Accesses (bits[127:64]) |
| 0b11 | 0b000 | 0b0010 | 0b0010 | 0b010 | APDBKeyLo_EL1 | Pointer Authentication Key B for Data Accesses (bits[63:0]) |
| 0b11 | 0b000 | 0b0010 | 0b0010 | 0b011 | APDBKeyHi_EL1 | Pointer Authentication Key B for Data Accesses (bits[127:64]) |
| 0b11 | 0b000 | 0b0010 | 0b0011 | 0b000 | APGAKeyLo_EL1 | Pointer Authentication Key A for Guest Accesses (bits[63:0]) |
| 0b11 | 0b000 | 0b0010 | 0b0011 | 0b001 | APGAKeyHi_EL1 | Pointer Authentication Key A for Guest Accesses (bits[127:64]) |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|-------------------------------|---|
| | | CRn | CRm | | | |
| 0b11 | 0b000 | 0b0100 | 0b0000 | 0b000 | SPSR_EL1 | Saved Program Status Register (EL1) |
| 0b11 | 0b000 | 0b0100 | 0b0000 | 0b001 | ELR_EL1 | Exception Link Register |
| 0b11 | 0b000 | 0b0100 | 0b0001 | 0b000 | SP_EL0 | Stack Pointer (EL0) |
| 0b11 | 0b000 | 0b0100 | 0b0010 | 0b000 | SPSel | Stack Pointer Select |
| 0b00 | 0b000 | 0b0100 | - | 0b101 | SPSel | Stack Pointer Select |
| 0b11 | 0b000 | 0b0100 | 0b0010 | 0b010 | CurrentEL | Current Exception Level |
| 0b11 | 0b000 | 0b0100 | 0b0010 | 0b011 | PAN | Privileged Access Never |
| 0b00 | 0b000 | 0b0100 | - | 0b100 | PAN | Privileged Access Never |
| 0b11 | 0b000 | 0b0100 | 0b0010 | 0b100 | UAO | User Access Override |
| 0b00 | 0b000 | 0b0100 | - | 0b011 | UAO | User Access Override |
| 0b11 | 0b000 | 0b0100 | 0b0011 | 0b000 | ALLINT | All Interrupt Mask Bit |
| 0b00 | 0b001 | 0b0100 | - | 0b000 | ALLINT | All Interrupt Mask Bit |
| 0b11 | 0b000 | 0b0100 | 0b0110 | 0b000 | ICC_PMR_EL1 | Interrupt Controller Priority Register |
| 0b11 | 0b000 | 0b0101 | 0b0001 | 0b000 | AFSR0_EL1 | Auxiliary Status Register 0 (EL1) |
| 0b11 | 0b000 | 0b0101 | 0b0001 | 0b001 | AFSR1_EL1 | Auxiliary Status Register 1 (EL1) |
| 0b11 | 0b000 | 0b0101 | 0b0010 | 0b000 | ESR_EL1 | Exception Syndrome Register |
| 0b11 | 0b000 | 0b0101 | 0b0011 | 0b000 | ERRIDR_EL1 | Error Register |
| 0b11 | 0b000 | 0b0101 | 0b0011 | 0b001 | ERRSELR_EL1 | Error Register Select |
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b000 | ERXFR_EL1 | Selected Record Identifier Register |
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b001 | ERXCTLR_EL1 | Selected Record Control Register |
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b010 | ERXSTATUS_EL1 | Selected Record Identifier Status Register |
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b011 | ERXADDR_EL1 | Selected Record Address Register |
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b100 | ERXPEGF_EL1 | Selected Pseudo-fault Generation Feature register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|------------------------------|--|
| | | CRn | CRm | | | |
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b101 | ERXPGCTL_EL1 | Selected Pseudo-fault Generator Control Register |
| 0b11 | 0b000 | 0b0101 | 0b0100 | 0b110 | ERXPGCDN_EL1 | Selected Pseudo-fault Generator Countdown register |
| 0b11 | 0b000 | 0b0101 | 0b0101 | 0b000 | ERXMISC0_EL1 | Selected Record Miscellaneous Register |
| 0b11 | 0b000 | 0b0101 | 0b0101 | 0b001 | ERXMISC1_EL1 | Selected Record Miscellaneous Register |
| 0b11 | 0b000 | 0b0101 | 0b0101 | 0b010 | ERXMISC2_EL1 | Selected Record Miscellaneous Register |
| 0b11 | 0b000 | 0b0101 | 0b0101 | 0b011 | ERXMISC3_EL1 | Selected Record Miscellaneous Register |
| 0b11 | 0b000 | 0b0101 | 0b0110 | 0b000 | TFSR_EL1 | Tag Fault Status Register (EL1) |
| 0b11 | 0b000 | 0b0101 | 0b0110 | 0b001 | TFSRE0_EL1 | Tag Fault Status Register (EL0). |
| 0b11 | 0b000 | 0b0110 | 0b0000 | 0b000 | FAR_EL1 | Fault Address Register |
| 0b11 | 0b000 | 0b0111 | 0b0100 | 0b000 | PAR_EL1 | Physical Address Register |
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b000 | PMSCR_EL1 | Statistic Profiling Control Register |
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b001 | PMSNEVFR_EL1 | Sampling Inverted Filter Register |
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b010 | PMSICR_EL1 | Sampling Interval Counter Register |
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b011 | PMSIRR_EL1 | Sampling Interval Register |
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b100 | PMSFCR_EL1 | Sampling Control Register |
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b101 | PMSEVFR_EL1 | Sampling Filter Register |
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b110 | PMSLATFR_EL1 | Sampling Latency Register |
| 0b11 | 0b000 | 0b1001 | 0b1001 | 0b111 | PMSIDR_EL1 | Sampling Profiling Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|--------------------------------|--|
| | | CRn | CRm | | | |
| 0b11 | 0b000 | 0b1001 | 0b1010 | 0b000 | PMBLIMITR_EL1 | Profiling Limit Address Register |
| 0b11 | 0b000 | 0b1001 | 0b1010 | 0b001 | PMBPTR_EL1 | Profiling Write Pointer Register |
| 0b11 | 0b000 | 0b1001 | 0b1010 | 0b011 | PMBSR_EL1 | Profiling Status/syndrom Register |
| 0b11 | 0b000 | 0b1001 | 0b1010 | 0b111 | PMBIDR_EL1 | Profiling ID Register |
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b000 | TRBLIMITR_EL1 | Trace Buffer Limit Address Register |
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b001 | TRBPTR_EL1 | Trace Buffer Write Pointer Register |
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b010 | TRBBASER_EL1 | Trace Buffer Base Address Register |
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b011 | TRBSR_EL1 | Trace Buffer Status/syndrom Register |
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b100 | TRBMAR_EL1 | Trace Buffer Memory Attribute Register |
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b110 | TRBTRG_EL1 | Trace Buffer Trigger Counter Register |
| 0b11 | 0b000 | 0b1001 | 0b1011 | 0b111 | TRBIDR_EL1 | Trace Buffer ID Register |
| 0b11 | 0b000 | 0b1001 | 0b1110 | 0b001 | PMINTENSET_EL1 | Performance Monitor: Interrupt Enable Set register |
| 0b11 | 0b000 | 0b1001 | 0b1110 | 0b010 | PMINTENCLR_EL1 | Performance Monitor: Interrupt Enable Clear register |
| 0b11 | 0b000 | 0b1001 | 0b1110 | 0b110 | PMMIR_EL1 | Performance Monitor: Machine Identification Register |
| 0b11 | 0b000 | 0b1010 | 0b0010 | 0b000 | MAIR_EL1 | Memory Attribute Indirect Register |
| 0b11 | 0b000 | 0b1010 | 0b0011 | 0b000 | AMAIR_EL1 | Auxiliary Memory Attribute Indirect Register |
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b000 | LORSA_EL1 | LORegion Address Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|------------|---------------------------------------|---|
| | | CRn | CRm | | | |
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b001 | LOREA_EL1 | LORegion Address |
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b010 | LORN_EL1 | LORegion Number |
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b011 | LORC_EL1 | LORegion Control |
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b100 | MPAMIDR_EL1 | MPAM Register |
| 0b11 | 0b000 | 0b1010 | 0b0100 | 0b111 | LORID_EL1 | LORegion (EL1) |
| 0b11 | 0b000 | 0b1010 | 0b0101 | 0b000 | MPAM1_EL1 | MPAM1 Register |
| 0b11 | 0b000 | 0b1010 | 0b0101 | 0b001 | MPAM0_EL1 | MPAM0 Register |
| 0b11 | 0b000 | 0b1010 | 0b0101 | 0b011 | MPAMSM_EL1 | MPAM Streaming Mode Register |
| 0b11 | 0b000 | 0b1100 | 0b0000 | 0b000 | VBAR_EL1 | Vector Base Address Register |
| 0b11 | 0b000 | 0b1100 | 0b0000 | 0b001 | RVBAR_EL1 | Reset Vector Base Address Register and EL3 implementation |
| 0b11 | 0b000 | 0b1100 | 0b0000 | 0b010 | RMR_EL1 | Reset Management Register |
| 0b11 | 0b000 | 0b1100 | 0b0001 | 0b000 | ISR_EL1 | Interrupt Status Register |
| 0b11 | 0b000 | 0b1100 | 0b0001 | 0b001 | DISR_EL1 | Deferred Interrupt Status Register |
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b000 | ICC_IAR0_EL1 | Interrupt Controller Interrupt Acknowledge Register |
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b001 | ICC_EOIR0_EL1 | Interrupt Controller Of Interrupt Register |
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b010 | ICC_HPPIR0_EL1 | Interrupt Controller Highest Pending Interrupt Register |
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b011 | ICC_BPR0_EL1 | Interrupt Controller Binary Pending Register |
| 0b11 | 0b000 | 0b1100 | 0b1000 | 0b1:n[1:0] | ICC_AP0R<n>_EL1 | Interrupt Controller Active Pending Group 0 Register |
| 0b11 | 0b000 | 0b1100 | 0b1001 | 0b0:n[1:0] | ICC_AP1R<n>_EL1 | Interrupt Controller Active Pending Group 1 Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|---------------------------------|--|
| | | CRn | CRm | | | |
| 0b11 | 0b000 | 0b1100 | 0b1001 | 0b101 | ICC_NMIAR1_EL1 | Interrupt Controller maskable Interrupt Acknowledge Register |
| 0b11 | 0b000 | 0b1100 | 0b1011 | 0b001 | ICC_DIR_EL1 | Interrupt Controller Deactivate Interrupt Register |
| 0b11 | 0b000 | 0b1100 | 0b1011 | 0b011 | ICC_RPR_EL1 | Interrupt Controller Running Priority Register |
| 0b11 | 0b000 | 0b1100 | 0b1011 | 0b101 | ICC_SGI1R_EL1 | Interrupt Controller Software Generated Interrupt 1 Register |
| 0b11 | 0b000 | 0b1100 | 0b1011 | 0b110 | ICC_ASGI1R_EL1 | Interrupt Controller Software Generated Interrupt 1 Register |
| 0b11 | 0b000 | 0b1100 | 0b1011 | 0b111 | ICC_SGI0R_EL1 | Interrupt Controller Software Generated Interrupt 0 Register |
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b000 | ICC_IAR1_EL1 | Interrupt Controller Interrupt Acknowledge Register |
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b001 | ICC_EOIR1_EL1 | Interrupt Controller End Of Interrupt Register |
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b010 | ICC_HPPIR1_EL1 | Interrupt Controller Highest Pending Interrupt Register |
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b011 | ICC_BPR1_EL1 | Interrupt Controller Binary Pending Register |
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b100 | ICC_CTLR_EL1 | Interrupt Controller Control Register |
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b101 | ICC_SRE_EL1 | Interrupt Controller System Register |
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b110 | ICC_IGRPEN0_EL1 | Interrupt Controller |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|---------------------------------|--|
| | | CRn | CRm | | | |
| | | | | | | Interrupt 0 Enable register |
| 0b11 | 0b000 | 0b1100 | 0b1100 | 0b111 | ICC_IGRPEN1_EL1 | Interrupt Controller Interrupt 1 Enable register |
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b001 | CONTEXTIDR_EL1 | Context Register |
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b100 | TPIDR_EL1 | EL1 Software Thread ID Register |
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b101 | ACCDATA_EL1 | Accelerator Data |
| 0b11 | 0b000 | 0b1101 | 0b0000 | 0b111 | SCXTNUM_EL1 | EL1 Realtime Software Context Number |
| 0b11 | 0b000 | 0b1110 | 0b0001 | 0b000 | CNTKCTL_EL1 | Counter Kernel Control register |
| 0b11 | 0b001 | 0b0000 | 0b0000 | 0b000 | CCSIDR_EL1 | Current Size ID Register |
| 0b11 | 0b001 | 0b0000 | 0b0000 | 0b001 | CLIDR_EL1 | Cache Level Register |
| 0b11 | 0b001 | 0b0000 | 0b0000 | 0b010 | CCSIDR2_EL1 | Current Size ID Register |
| 0b11 | 0b001 | 0b0000 | 0b0000 | 0b100 | GMID_EL1 | Multiple transfer register |
| 0b11 | 0b001 | 0b0000 | 0b0000 | 0b110 | SMIDR_EL1 | Streaming Mode Identification Register |
| 0b11 | 0b001 | 0b0000 | 0b0000 | 0b111 | AIDR_EL1 | Auxiliary Register |
| 0b11 | 0b010 | 0b0000 | 0b0000 | 0b000 | CSSELR_EL1 | Cache Selection Register |
| 0b11 | 0b011 | 0b0000 | 0b0000 | 0b001 | CTR_EL0 | Cache Type Register |
| 0b11 | 0b011 | 0b0000 | 0b0000 | 0b111 | DCZID_EL0 | Data Cache Zero ID |
| 0b11 | 0b011 | 0b0010 | 0b0100 | 0b000 | RNDR | Random Number |
| 0b11 | 0b011 | 0b0010 | 0b0100 | 0b001 | RNDRRS | Reseeded Random Number |
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b000 | NZCV | Condition |
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b001 | DAIF | Interrupt Bits |
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b010 | SVCR | Streaming Vector Control Register |
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b101 | DIT | Data Independent Timing |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|--------------------------------|---|
| | | CRn | CRm | | | |
| 0b00 | 0b011 | 0b0100 | - | 0b010 | DIT | Data Independent Timing |
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b110 | SSBS | Speculative Store By Safe |
| 0b00 | 0b011 | 0b0100 | - | 0b001 | SSBS | Speculative Store By Safe |
| 0b11 | 0b011 | 0b0100 | 0b0010 | 0b111 | TCO | Tag Check Override |
| 0b00 | 0b011 | 0b0100 | - | 0b100 | TCO | Tag Check Override |
| 0b11 | 0b011 | 0b0100 | 0b0100 | 0b000 | FPCR | Floating Control Register |
| 0b11 | 0b011 | 0b0100 | 0b0100 | 0b001 | FPSR | Floating Status Register |
| 0b11 | 0b011 | 0b0100 | 0b0101 | 0b000 | DSPSR_EL0 | Debug Status Program Register |
| 0b11 | 0b011 | 0b0100 | 0b0101 | 0b001 | DLR_EL0 | Debug Link Register |
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b000 | PMCRL_EL0 | Performance Monitor Control Register |
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b001 | PMCNTENSET_EL0 | Performance Monitor Enable S register |
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b010 | PMCNTENCLR_EL0 | Performance Monitor Enable C register |
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b011 | PMOVSLR_EL0 | Performance Monitor Overflow Status C Register |
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b100 | PMSWINC_EL0 | Performance Monitor Software Increment register |
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b101 | PMSELR_EL0 | Performance Monitor Counter Selection Register |
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b110 | PMCEID0_EL0 | Performance Monitor Common Identifier register |
| 0b11 | 0b011 | 0b1001 | 0b1100 | 0b111 | PMCEID1_EL0 | Performance Monitor Common Identifier register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|---------------------------------|--|
| | | CRn | CRm | | | |
| 0b11 | 0b011 | 0b1001 | 0b1101 | 0b000 | PMCCNTR_EL0 | Performance Monitor: Count Register |
| 0b11 | 0b011 | 0b1001 | 0b1101 | 0b001 | PMXEVTYPER_EL0 | Performance Monitor: Selected Type Register |
| 0b11 | 0b011 | 0b1001 | 0b1101 | 0b010 | PMXEVCNTR_EL0 | Performance Monitor: Selected Count Register |
| 0b11 | 0b011 | 0b1001 | 0b1110 | 0b000 | PMUSERENR_EL0 | Performance Monitor: Enable Filter |
| 0b11 | 0b011 | 0b1001 | 0b1110 | 0b011 | PMOVSSET_EL0 | Performance Monitor: Overflow Status Register |
| 0b11 | 0b011 | 0b1101 | 0b0000 | 0b010 | TPIDR_EL0 | EL0 Thread ID Register |
| 0b11 | 0b011 | 0b1101 | 0b0000 | 0b011 | TPIDRRO_EL0 | EL0 Thread ID Register |
| 0b11 | 0b011 | 0b1101 | 0b0000 | 0b101 | TPIDR2_EL0 | EL0 Thread ID Register |
| 0b11 | 0b011 | 0b1101 | 0b0000 | 0b111 | SCXTNUM_EL0 | EL0 Software Context Number |
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b000 | AMCR_EL0 | Activity Monitor: Control Register |
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b001 | AMCFGR_EL0 | Activity Monitor: Configuration Register |
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b010 | AMCGCR_EL0 | Activity Monitor: Counter Configuration Register |
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b011 | AMUSERENR_EL0 | Activity Monitor: Enable Filter |
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b100 | AMCNTENCLR0_EL0 | Activity Monitor: Enable Counter Register |
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b101 | AMCNTENSET0_EL0 | Activity Monitor: Enable Counter Register |
| 0b11 | 0b011 | 0b1101 | 0b0010 | 0b110 | AMCG1IDR_EL0 | Activity Monitor: Counter Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|------------|--------|---|--|
| | | CRn | CRm | | | |
| | | | | | | 1 Identifier Register |
| 0b11 | 0b011 | 0b1101 | 0b0011 | 0b000 | AMCNTENCLR1_EL0 | Activity Monitor: Enable Counter Register |
| 0b11 | 0b011 | 0b1101 | 0b0011 | 0b001 | AMCNTENSET1_EL0 | Activity Monitor: Enable Set Register |
| 0b11 | 0b011 | 0b1101 | 0b010:n[3] | n[2:0] | AMEVCNTR0<n>_EL0 | Activity Monitor: Counter Register |
| 0b11 | 0b011 | 0b1101 | 0b011:n[3] | n[2:0] | AMEVTYPEPER0<n>_EL0 | Activity Monitor: Type Register 0 |
| 0b11 | 0b011 | 0b1101 | 0b110:n[3] | n[2:0] | AMEVCNTR1<n>_EL0 | Activity Monitor: Counter Register |
| 0b11 | 0b011 | 0b1101 | 0b111:n[3] | n[2:0] | AMEVTYPEPER1<n>_EL0 | Activity Monitor: Type Register 1 |
| 0b11 | 0b011 | 0b1110 | 0b0000 | 0b000 | CNTFRQ_EL0 | Counter: Frequency register |
| 0b11 | 0b011 | 0b1110 | 0b0000 | 0b001 | CNTPCT_EL0 | Counter: Physical register |
| 0b11 | 0b011 | 0b1110 | 0b0000 | 0b010 | CNTVCT_EL0 | Counter: Virtual Counter register |
| 0b11 | 0b011 | 0b1110 | 0b0000 | 0b101 | CNTPCTSS_EL0 | Counter: Self-Synchronous Physical register |
| 0b11 | 0b011 | 0b1110 | 0b0000 | 0b110 | CNTVCTSS_EL0 | Counter: Self-Synchronous Virtual Counter register |
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b000 | CNTP_TVAL_EL0 | Counter: Physical Timer Value register |
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b001 | CNTP_CTL_EL0 | Counter: Physical Control register |
| 0b11 | 0b011 | 0b1110 | 0b0010 | 0b010 | CNTP_CVAL_EL0 | Counter: Physical Compare register |
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b000 | CNTV_TVAL_EL0 | Counter: Virtual Timer Value register |
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b001 | CNTV_CTL_EL0 | Counter: Virtual Timer Control register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|-------------|--------|--|---|
| | | CRn | CRm | | | |
| 0b11 | 0b011 | 0b1110 | 0b0011 | 0b010 | CNTV_CVAL_EL0 | Counter Virtual Time Compare register |
| 0b11 | 0b011 | 0b1110 | 0b10:n[4:3] | n[2:0] | PMEVCNTR<n>_EL0 | Performance Monitor Count Register |
| 0b11 | 0b011 | 0b1110 | 0b1111 | 0b111 | PMCCFILTR_EL0 | Performance Monitor Count Filter Register |
| 0b11 | 0b011 | 0b1110 | 0b11:n[4:3] | n[2:0] | PMEVTYPER<n>_EL0 | Performance Monitor Type Register |
| 0b11 | 0b100 | 0b0000 | 0b0000 | 0b000 | VPIDR_EL2 | Virtualization Process ID Register |
| 0b11 | 0b100 | 0b0000 | 0b0000 | 0b101 | VMPIDR_EL2 | Virtualization Multiprocessor ID Register |
| 0b11 | 0b100 | 0b0001 | 0b0000 | 0b000 | SCTLR_EL2 | System Control Register |
| 0b11 | 0b100 | 0b0001 | 0b0000 | 0b001 | ACTLR_EL2 | Auxiliary Control Register |
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b000 | HCR_EL2 | Hypervisor Configuration Register |
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b001 | MDCR_EL2 | Monitor Configuration Register |
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b010 | CPTR_EL2 | Architectural Feature Register |
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b011 | HSTR_EL2 | Hypervisor System Trap Register |
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b100 | HFGRTR_EL2 | Hypervisor Fine-Grained Read Trap Register |
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b101 | HFGWTR_EL2 | Hypervisor Fine-Grained Write Trap Register |
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b110 | HFGITR_EL2 | Hypervisor Fine-Grained Instruction Trap Register |
| 0b11 | 0b100 | 0b0001 | 0b0001 | 0b111 | HACR_EL2 | Hypervisor Auxiliary Control Register |
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b000 | ZCR_EL2 | SVE Control Register |
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b001 | TRFCR_EL2 | Trace Filter Control Register |
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b010 | HCRX_EL2 | Extended Hypervisor Configuration Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|------------------------------|---|
| | | CRn | CRm | | | |
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b101 | SMPRIMAP_EL2 | Streaming Mode Privilege Mapping Register |
| 0b11 | 0b100 | 0b0001 | 0b0010 | 0b110 | SMCR_EL2 | SME Control Register |
| 0b11 | 0b100 | 0b0001 | 0b0011 | 0b001 | SDER32_EL2 | AArch32 Secure I/O Enable Register |
| 0b11 | 0b100 | 0b0010 | 0b0000 | 0b000 | TTBR0_EL2 | Translation Table Base Register (EL2) |
| 0b11 | 0b100 | 0b0010 | 0b0000 | 0b001 | TTBR1_EL2 | Translation Table Base Register (EL2) |
| 0b11 | 0b100 | 0b0010 | 0b0000 | 0b010 | TCR_EL2 | Translation Control Register |
| 0b11 | 0b100 | 0b0010 | 0b0001 | 0b000 | VTTBR_EL2 | Virtualization Translation Table Base Register |
| 0b11 | 0b100 | 0b0010 | 0b0001 | 0b010 | VTCR_EL2 | Virtualization Translation Control Register |
| 0b11 | 0b100 | 0b0010 | 0b0010 | 0b000 | VNCR_EL2 | Virtualization Control Register |
| 0b11 | 0b100 | 0b0010 | 0b0110 | 0b000 | VSTTBR_EL2 | Virtualization Secure Translation Table Base Register |
| 0b11 | 0b100 | 0b0010 | 0b0110 | 0b010 | VSTCR_EL2 | Virtualization Secure Translation Control Register |
| 0b11 | 0b100 | 0b0011 | 0b0000 | 0b000 | DACR32_EL2 | Domain Control Register |
| 0b11 | 0b100 | 0b0011 | 0b0001 | 0b100 | HDFGRTR_EL2 | Hypervisor Debug Free-Grained Trap Register |
| 0b11 | 0b100 | 0b0011 | 0b0001 | 0b101 | HDFGWTR_EL2 | Hypervisor Debug Free-Grained Trap Register |
| 0b11 | 0b100 | 0b0011 | 0b0001 | 0b110 | HAFGRTR_EL2 | Hypervisor Activity Monitor-Grained Trap Register |
| 0b11 | 0b100 | 0b0100 | 0b0000 | 0b000 | SPSR_EL2 | Saved Program Status Register (EL2) |
| 0b11 | 0b100 | 0b0100 | 0b0000 | 0b001 | ELR_EL2 | Exception Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|------------------------------|---|
| | | CRn | CRm | | | |
| 0b11 | 0b100 | 0b0100 | 0b0001 | 0b000 | SP_EL1 | Stack Pointer (EL1) |
| 0b11 | 0b100 | 0b0100 | 0b0011 | 0b000 | SPSR_irq | Saved Program Status Register (IRQ mode) |
| 0b11 | 0b100 | 0b0100 | 0b0011 | 0b001 | SPSR_abt | Saved Program Status Register (Abort mode) |
| 0b11 | 0b100 | 0b0100 | 0b0011 | 0b010 | SPSR_und | Saved Program Status Register (Undefined mode) |
| 0b11 | 0b100 | 0b0100 | 0b0011 | 0b011 | SPSR_fiq | Saved Program Status Register (FIQ mode) |
| 0b11 | 0b100 | 0b0101 | 0b0000 | 0b001 | IFSR32_EL2 | Instruction Fault Status Register |
| 0b11 | 0b100 | 0b0101 | 0b0001 | 0b000 | AFSR0_EL2 | Auxiliary Status Register 0 (EL2) |
| 0b11 | 0b100 | 0b0101 | 0b0001 | 0b001 | AFSR1_EL2 | Auxiliary Status Register 1 (EL2) |
| 0b11 | 0b100 | 0b0101 | 0b0010 | 0b000 | ESR_EL2 | Exception Syndrome Register |
| 0b11 | 0b100 | 0b0101 | 0b0010 | 0b011 | VSESR_EL2 | Virtual Exception Syndrome Register |
| 0b11 | 0b100 | 0b0101 | 0b0011 | 0b000 | FPEXC32_EL2 | Floating-Point Exception Control Register |
| 0b11 | 0b100 | 0b0101 | 0b0110 | 0b000 | TFSR_EL2 | Tag Fault Status Register (EL2) |
| 0b11 | 0b100 | 0b0110 | 0b0000 | 0b000 | FAR_EL2 | Fault Address Register |
| 0b11 | 0b100 | 0b0110 | 0b0000 | 0b100 | HPFAR_EL2 | Hypervisor Fault Address Register |
| 0b11 | 0b100 | 0b1001 | 0b1001 | 0b000 | PMSCR_EL2 | Statistical Profiling Control Register |
| 0b11 | 0b100 | 0b1010 | 0b0010 | 0b000 | MAIR_EL2 | Memory Attribute Indirection Register |
| 0b11 | 0b100 | 0b1010 | 0b0011 | 0b000 | AMAIR_EL2 | Auxiliary Memory Attribute Indirection Register |
| 0b11 | 0b100 | 0b1010 | 0b0100 | 0b000 | MPAMHCR_EL2 | MPAM Hypervisor Control Register |
| 0b11 | 0b100 | 0b1010 | 0b0100 | 0b001 | MPAMVPMV_EL2 | MPAM Virtual Partition |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|------------|---------------------------------------|---|
| | | CRn | CRm | | | |
| | | | | | | Mapping Register |
| 0b11 | 0b100 | 0b1010 | 0b0101 | 0b000 | MPAM2_EL2 | MPAM2 Register |
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b000 | MPAMVPM0_EL2 | MPAM VPARTID Mapping Register |
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b001 | MPAMVPM1_EL2 | MPAM VPARTID Mapping Register |
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b010 | MPAMVPM2_EL2 | MPAM VPARTID Mapping Register |
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b011 | MPAMVPM3_EL2 | MPAM VPARTID Mapping Register |
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b100 | MPAMVPM4_EL2 | MPAM VPARTID Mapping Register |
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b101 | MPAMVPM5_EL2 | MPAM VPARTID Mapping Register |
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b110 | MPAMVPM6_EL2 | MPAM VPARTID Mapping Register |
| 0b11 | 0b100 | 0b1010 | 0b0110 | 0b111 | MPAMVPM7_EL2 | MPAM VPARTID Mapping Register |
| 0b11 | 0b100 | 0b1100 | 0b0000 | 0b000 | VBAR_EL2 | Vector Base Address Register |
| 0b11 | 0b100 | 0b1100 | 0b0000 | 0b001 | RVBAR_EL2 | Reset Vector Base Address Register, not implemented |
| 0b11 | 0b100 | 0b1100 | 0b0000 | 0b010 | RMR_EL2 | Reset Management Register |
| 0b11 | 0b100 | 0b1100 | 0b0001 | 0b001 | VDISR_EL2 | Virtual Deferred Interrupt Status Register |
| 0b11 | 0b100 | 0b1100 | 0b1000 | 0b0:n[1:0] | ICH_AP0R<n>_EL2 | Interrupt Controller Active Period Group 0 Register |
| 0b11 | 0b100 | 0b1100 | 0b1001 | 0b0:n[1:0] | ICH_AP1R<n>_EL2 | Interrupt Controller Active Period Group 1 Register |
| 0b11 | 0b100 | 0b1100 | 0b1001 | 0b101 | ICC_SRE_EL2 | Interrupt Controller |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|------------|--------|---|---|
| | | CRn | CRm | | | |
| | | | | | | System Register register |
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b000 | ICH_HCR_EL2 | Interrupt Controller Control Register |
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b001 | ICH_VTR_EL2 | Interrupt Controller Type Register |
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b010 | ICH_MISR_EL2 | Interrupt Controller Maintenance Interrupt Register |
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b011 | ICH_EISR_EL2 | Interrupt Controller of Interrupt Status Register |
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b101 | ICH_ELRSR_EL2 | Interrupt Controller Empty List Register |
| 0b11 | 0b100 | 0b1100 | 0b1011 | 0b111 | ICH_VMCR_EL2 | Interrupt Controller Virtual Machine Control Register |
| 0b11 | 0b100 | 0b1100 | 0b110:n[3] | n[2:0] | ICH_LR<n>_EL2 | Interrupt Controller Register |
| 0b11 | 0b100 | 0b1101 | 0b0000 | 0b001 | CONTEXTIDR_EL2 | Context Register |
| 0b11 | 0b100 | 0b1101 | 0b0000 | 0b010 | TPIDR_EL2 | EL2 Software Thread ID Register |
| 0b11 | 0b100 | 0b1101 | 0b0000 | 0b111 | SCXTNUM_EL2 | EL2 Realtime Software Context Number |
| 0b11 | 0b100 | 0b1101 | 0b100:n[3] | n[2:0] | AMEVCNTVOFF0<n>_EL2 | Activity Monitor Counter Offset Register 0 |
| 0b11 | 0b100 | 0b1101 | 0b101:n[3] | n[2:0] | AMEVCNTVOFF1<n>_EL2 | Activity Monitor Counter Offset Register 1 |
| 0b11 | 0b100 | 0b1110 | 0b0000 | 0b011 | CNTVOFF_EL2 | Counter Virtual Control register |
| 0b11 | 0b100 | 0b1110 | 0b0000 | 0b110 | CNTPOFF_EL2 | Counter Physical Control register |
| 0b11 | 0b100 | 0b1110 | 0b0001 | 0b000 | CNTHCTL_EL2 | Counter Hypervisor Control Register |
| 0b11 | 0b100 | 0b1110 | 0b0010 | 0b000 | CNTHP_TVAL_EL2 | Counter Physical Control Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|---------------------------------|---|
| | | CRn | CRm | | | |
| | | | | | | TimerVa register |
| 0b11 | 0b100 | 0b1110 | 0b0010 | 0b001 | CNTHP_CTL_EL2 | Counter Hypervis Physical Control |
| 0b11 | 0b100 | 0b1110 | 0b0010 | 0b010 | CNTHP_CVAL_EL2 | Counter Physical Compare register |
| 0b11 | 0b100 | 0b1110 | 0b0011 | 0b000 | CNTHV_TVAL_EL2 | Counter Virtual T TimerVa Register |
| 0b11 | 0b100 | 0b1110 | 0b0011 | 0b001 | CNTHV_CTL_EL2 | Counter Virtual T Control (EL2) |
| 0b11 | 0b100 | 0b1110 | 0b0011 | 0b010 | CNTHV_CVAL_EL2 | Counter Virtual T Compare register |
| 0b11 | 0b100 | 0b1110 | 0b0100 | 0b000 | CNTHVS_TVAL_EL2 | Counter Secure V Timer TimerVa register |
| 0b11 | 0b100 | 0b1110 | 0b0100 | 0b001 | CNTHVS_CTL_EL2 | Counter Secure V Timer C register |
| 0b11 | 0b100 | 0b1110 | 0b0100 | 0b010 | CNTHVS_CVAL_EL2 | Counter Secure V Timer Compare register |
| 0b11 | 0b100 | 0b1110 | 0b0101 | 0b000 | CNTHPS_TVAL_EL2 | Counter Secure P Timer TimerVa register |
| 0b11 | 0b100 | 0b1110 | 0b0101 | 0b001 | CNTHPS_CTL_EL2 | Counter Secure P Timer C register |
| 0b11 | 0b100 | 0b1110 | 0b0101 | 0b010 | CNTHPS_CVAL_EL2 | Counter Secure P Timer Compare register |
| 0b11 | 0b110 | 0b0001 | 0b0000 | 0b000 | SCTLR_EL3 | System C Register |
| 0b11 | 0b110 | 0b0001 | 0b0000 | 0b001 | ACTLR_EL3 | Auxiliary Control Register |
| 0b11 | 0b110 | 0b0001 | 0b0001 | 0b000 | SCR_EL3 | Secure Configur Register |
| 0b11 | 0b110 | 0b0001 | 0b0001 | 0b001 | SDER32_EL3 | AArch32 Secure I Enable F |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|---------------------------|---|
| | | CRn | CRm | | | |
| 0b11 | 0b110 | 0b0001 | 0b0001 | 0b010 | CPTR_EL3 | Architectural Feature Register |
| 0b11 | 0b110 | 0b0001 | 0b0010 | 0b000 | ZCR_EL3 | SVE Control Register |
| 0b11 | 0b110 | 0b0001 | 0b0010 | 0b110 | SMCR_EL3 | SME Control Register |
| 0b11 | 0b110 | 0b0001 | 0b0011 | 0b001 | MDCR_EL3 | Monitor Configuration Register |
| 0b11 | 0b110 | 0b0010 | 0b0000 | 0b000 | TTBR0_EL3 | Translation Table Base Register (EL3) |
| 0b11 | 0b110 | 0b0010 | 0b0000 | 0b010 | TCR_EL3 | Translation Control Register |
| 0b11 | 0b110 | 0b0010 | 0b0001 | 0b100 | GPTBR_EL3 | Granule Protection Table Base Register |
| 0b11 | 0b110 | 0b0010 | 0b0001 | 0b110 | GPCCR_EL3 | Granule Protection Check Control Register |
| 0b11 | 0b110 | 0b0100 | 0b0000 | 0b000 | SPSR_EL3 | Saved Program Status Register (EL3) |
| 0b11 | 0b110 | 0b0100 | 0b0000 | 0b001 | ELR_EL3 | Exception Link Register |
| 0b11 | 0b110 | 0b0100 | 0b0001 | 0b000 | SP_EL2 | Stack Pointer (EL2) |
| 0b11 | 0b110 | 0b0101 | 0b0001 | 0b000 | AFSR0_EL3 | Auxiliary Status Register 0 (EL3) |
| 0b11 | 0b110 | 0b0101 | 0b0001 | 0b001 | AFSR1_EL3 | Auxiliary Status Register 1 (EL3) |
| 0b11 | 0b110 | 0b0101 | 0b0010 | 0b000 | ESR_EL3 | Exception Syndrome Register |
| 0b11 | 0b110 | 0b0101 | 0b0110 | 0b000 | TFSR_EL3 | Tag Fault Status Register (EL3) |
| 0b11 | 0b110 | 0b0110 | 0b0000 | 0b000 | FAR_EL3 | Fault Address Register |
| 0b11 | 0b110 | 0b0110 | 0b0000 | 0b101 | MFAR_EL3 | PA Fault Address Register |
| 0b11 | 0b110 | 0b1010 | 0b0010 | 0b000 | MAIR_EL3 | Memory Attribute Indirection Register |
| 0b11 | 0b110 | 0b1010 | 0b0011 | 0b000 | AMAIR_EL3 | Auxiliary Memory Attribute Indirection Register |
| 0b11 | 0b110 | 0b1010 | 0b0101 | 0b000 | MPAM3_EL3 | MPAM3 Register |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|---------------------------------|---|
| | | CRn | CRm | | | |
| 0b11 | 0b110 | 0b1100 | 0b0000 | 0b000 | VBAR_EL3 | Vector Base Address Register |
| 0b11 | 0b110 | 0b1100 | 0b0000 | 0b001 | RVBAR_EL3 | Reset Vector Base Address Register implementation |
| 0b11 | 0b110 | 0b1100 | 0b0000 | 0b010 | RMR_EL3 | Reset Management Register |
| 0b11 | 0b110 | 0b1100 | 0b1100 | 0b100 | ICC_CTLR_EL3 | Interrupt Controller Control Register |
| 0b11 | 0b110 | 0b1100 | 0b1100 | 0b101 | ICC_SRE_EL3 | Interrupt Controller System Register |
| 0b11 | 0b110 | 0b1100 | 0b1100 | 0b111 | ICC_IGRPEN1_EL3 | Interrupt Controller Interrupt 1 Enable register |
| 0b11 | 0b110 | 0b1101 | 0b0000 | 0b010 | TPIDR_EL3 | EL3 Software Thread ID Register |
| 0b11 | 0b110 | 0b1101 | 0b0000 | 0b111 | SCXTNUM_EL3 | EL3 Realtime Software Context Number |
| 0b11 | 0b111 | 0b1110 | 0b0010 | 0b000 | CNTPS_TVAL_EL1 | Counter Physical Timer TimerValue register |
| 0b11 | 0b111 | 0b1110 | 0b0010 | 0b001 | CNTPS_CTL_EL1 | Counter Physical Timer Control register |
| 0b11 | 0b111 | 0b1110 | 0b0010 | 0b010 | CNTPS_CVAL_EL1 | Counter Physical Timer Compare register |

Accessed using TLBI:

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|-----------------------------------|--|
| | | CRn | CRm | | | |
| 0b01 | 0b000 | 0b1000 | 0b0001 | 0b000 | TLBI VMALLE1OS | TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0001 | 0b000 | TLBI VMALLE1OSNXS | TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0001 | 0b001 | TLBI VAE1OS | TLB Invalidate by VA, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0001 | 0b001 | TLBI VAE1OSNXS | TLB Invalidate by VA, EL1, Outer Shareable |

| Register selectors | | | | | Name | Description |
|--------------------|-------|--------|--------|-------|-----------------------------------|--|
| op0 | op1 | CRn | CRm | op2 | | |
| 0b01 | 0b000 | 0b1000 | 0b0001 | 0b010 | TLBI ASIDE1OS | TLB Invalidate by ASID, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0001 | 0b010 | TLBI ASIDE1OSNXS | TLB Invalidate by ASID, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0001 | 0b011 | TLBI VAAE1OS | TLB Invalidate by VA, All ASID, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0001 | 0b011 | TLBI VAAE1OSNXS | TLB Invalidate by VA, All ASID, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0001 | 0b101 | TLBI VALE1OS | TLB Invalidate by VA, Last level, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0001 | 0b101 | TLBI VALE1OSNXS | TLB Invalidate by VA, Last level, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0001 | 0b111 | TLBI VAALE1OS | TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0001 | 0b111 | TLBI VAALE1OSNXS | TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0010 | 0b001 | TLBI RVAE1IS | TLB Range Invalidate by VA, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0010 | 0b001 | TLBI RVAE1ISNXS | TLB Range Invalidate by VA, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0010 | 0b011 | TLBI RVAAE1IS | TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0010 | 0b011 | TLBI RVAAE1ISNXS | TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0010 | 0b101 | TLBI RVALE1IS | TLB Range Invalidate by VA, Last level, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0010 | 0b101 | TLBI RVALE1ISNXS | TLB Range Invalidate by VA, Last level, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0010 | 0b111 | TLBI RVAALE1IS | TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0010 | 0b111 | TLBI RVAALE1ISNXS | TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0011 | 0b000 | TLBI VMALLE1IS | TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0011 | 0b000 | TLBI VMALLE1ISNXS | TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0011 | 0b001 | TLBI VAE1IS | TLB Invalidate by VA, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0011 | 0b001 | TLBI VAE1ISNXS | TLB Invalidate by VA, EL1, Inner Shareable |

| Register selectors | | | | | Name | Description |
|--------------------|-------|--------|--------|-------|------------------------------------|--|
| op0 | op1 | CRn | CRm | op2 | | |
| 0b01 | 0b000 | 0b1000 | 0b0011 | 0b010 | TLBI ASIDE1IS | TLB Invalidate by ASID, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0011 | 0b010 | TLBI ASIDE1ISNXS | TLB Invalidate by ASID, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0011 | 0b011 | TLBI VAAE1IS | TLB Invalidate by VA, All ASID, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0011 | 0b011 | TLBI VAAE1ISNXS | TLB Invalidate by VA, All ASID, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0011 | 0b101 | TLBI VALE1IS | TLB Invalidate by VA, Last level, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0011 | 0b101 | TLBI VALE1ISNXS | TLB Invalidate by VA, Last level, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0011 | 0b111 | TLBI VAALE1IS | TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0011 | 0b111 | TLBI VAALE1ISNXS | TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0101 | 0b001 | TLBI RVAE1IOS | TLB Range Invalidate by VA, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0101 | 0b001 | TLBI RVAE1IOSNXS | TLB Range Invalidate by VA, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0101 | 0b011 | TLBI RVAAE1IOS | TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0101 | 0b011 | TLBI RVAAE1IOSNXS | TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0101 | 0b101 | TLBI RVALE1IOS | TLB Range Invalidate by VA, Last level, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0101 | 0b101 | TLBI RVALE1IOSNXS | TLB Range Invalidate by VA, Last level, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0101 | 0b111 | TLBI RVAALE1IOS | TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1001 | 0b0101 | 0b111 | TLBI RVAALE1IOSNXS | TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable |
| 0b01 | 0b000 | 0b1000 | 0b0110 | 0b001 | TLBI RVAE1 | TLB Range Invalidate by VA, EL1 |
| 0b01 | 0b000 | 0b1001 | 0b0110 | 0b001 | TLBI RVAE1NXS | TLB Range Invalidate by VA, EL1 |
| 0b01 | 0b000 | 0b1000 | 0b0110 | 0b011 | TLBI RVAAE1 | TLB Range Invalidate by VA, All ASID, EL1 |
| 0b01 | 0b000 | 0b1001 | 0b0110 | 0b011 | TLBI RVAAE1NXS | TLB Range Invalidate by VA, All ASID, EL1 |
| 0b01 | 0b000 | 0b1000 | 0b0110 | 0b101 | TLBI RVALE1 | TLB Range Invalidate by VA, Last level, EL1 |

| Register selectors | | | | | Name | Description |
|--------------------|-------|--------|--------|-------|------------------------------------|--|
| op0 | op1 | CRn | CRm | op2 | | |
| 0b01 | 0b000 | 0b1001 | 0b0110 | 0b101 | TLBI RVALE1NXS | TLB Range Invalidate by VA, Last level, EL1 |
| 0b01 | 0b000 | 0b1000 | 0b0110 | 0b111 | TLBI RVAALE1 | TLB Range Invalidate by VA, All ASID, Last level, EL1 |
| 0b01 | 0b000 | 0b1001 | 0b0110 | 0b111 | TLBI RVAALE1NXS | TLB Range Invalidate by VA, All ASID, Last level, EL1 |
| 0b01 | 0b000 | 0b1000 | 0b0111 | 0b000 | TLBI VMALLE1 | TLB Invalidate by VMID, All at stage 1, EL1 |
| 0b01 | 0b000 | 0b1001 | 0b0111 | 0b000 | TLBI VMALLE1NXS | TLB Invalidate by VMID, All at stage 1, EL1 |
| 0b01 | 0b000 | 0b1000 | 0b0111 | 0b001 | TLBI VAE1 | TLB Invalidate by VA, EL1 |
| 0b01 | 0b000 | 0b1001 | 0b0111 | 0b001 | TLBI VAE1NXS | TLB Invalidate by VA, EL1 |
| 0b01 | 0b000 | 0b1000 | 0b0111 | 0b010 | TLBI ASIDE1 | TLB Invalidate by ASID, EL1 |
| 0b01 | 0b000 | 0b1001 | 0b0111 | 0b010 | TLBI ASIDE1NXS | TLB Invalidate by ASID, EL1 |
| 0b01 | 0b000 | 0b1000 | 0b0111 | 0b011 | TLBI VAAE1 | TLB Invalidate by VA, All ASID, EL1 |
| 0b01 | 0b000 | 0b1001 | 0b0111 | 0b011 | TLBI VAAE1NXS | TLB Invalidate by VA, All ASID, EL1 |
| 0b01 | 0b000 | 0b1000 | 0b0111 | 0b101 | TLBI VALE1 | TLB Invalidate by VA, Last level, EL1 |
| 0b01 | 0b000 | 0b1001 | 0b0111 | 0b101 | TLBI VALE1NXS | TLB Invalidate by VA, Last level, EL1 |
| 0b01 | 0b000 | 0b1000 | 0b0111 | 0b111 | TLBI VAALE1 | TLB Invalidate by VA, All ASID, Last level, EL1 |
| 0b01 | 0b000 | 0b1001 | 0b0111 | 0b111 | TLBI VAALE1NXS | TLB Invalidate by VA, All ASID, Last level, EL1 |
| 0b01 | 0b100 | 0b1000 | 0b0000 | 0b001 | TLBI IPAS2E1IS | TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0000 | 0b001 | TLBI IPAS2E1ISNXS | TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0000 | 0b010 | TLBI RIPAS2E1IS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0000 | 0b010 | TLBI RIPAS2E1ISNXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0000 | 0b101 | TLBI IPAS2LE1IS | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0000 | 0b101 | TLBI IPAS2LE1ISNXS | TLB Invalidate by Intermediate Physical |

| op0 | op1 | Register selectors | | op2 | Name | Description |
|------|-------|--------------------|--------|-------|--------------------------------------|--|
| | | CRn | CRm | | | |
| | | | | | | Address, Stage 2, Last level, EL1, Inner Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0000 | 0b110 | TLBI RIPAS2LE1IS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0000 | 0b110 | TLBI RIPAS2LE1ISNXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0001 | 0b000 | TLBI ALLE2OS | TLB Invalidate All, EL2, Outer Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0001 | 0b000 | TLBI ALLE2OSNXS | TLB Invalidate All, EL2, Outer Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0001 | 0b001 | TLBI VAE2OS | TLB Invalidate by VA, EL2, Outer Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0001 | 0b001 | TLBI VAE2OSNXS | TLB Invalidate by VA, EL2, Outer Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0001 | 0b100 | TLBI ALLE1OS | TLB Invalidate All, EL1, Outer Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0001 | 0b100 | TLBI ALLE1OSNXS | TLB Invalidate All, EL1, Outer Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0001 | 0b101 | TLBI VALE2OS | TLB Invalidate by VA, Last level, EL2, Outer Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0001 | 0b101 | TLBI VALE2OSNXS | TLB Invalidate by VA, Last level, EL2, Outer Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0001 | 0b110 | TLBI VMALLS12E1OS | TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0001 | 0b110 | TLBI VMALLS12E1OSNXS | TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0010 | 0b001 | TLBI RVAE2IS | TLB Range Invalidate by VA, EL2, Inner Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0010 | 0b001 | TLBI RVAE2ISNXS | TLB Range Invalidate by VA, EL2, Inner Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0010 | 0b101 | TLBI RVALE2IS | TLB Range Invalidate by VA, Last level, EL2, Inner Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0010 | 0b101 | TLBI RVALE2ISNXS | TLB Range Invalidate by VA, Last level, EL2, Inner Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0011 | 0b000 | TLBI ALLE2IS | TLB Invalidate All, EL2, Inner Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0011 | 0b000 | TLBI ALLE2ISNXS | TLB Invalidate All, EL2, Inner Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0011 | 0b001 | TLBI VAE2IS | TLB Invalidate by VA, EL2, Inner Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0011 | 0b001 | TLBI VAE2ISNXS | TLB Invalidate by VA, EL2, Inner Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0011 | 0b100 | TLBI ALLE1IS | TLB Invalidate All, EL1, Inner Shareable |

| Register selectors | | | | | Name | Description |
|--------------------|-------|--------|--------|-------|--------------------------------------|--|
| op0 | op1 | CRn | CRm | op2 | | |
| 0b01 | 0b100 | 0b1001 | 0b0011 | 0b100 | TLBI ALLE1ISNXS | TLB Invalidate All, EL1, Inner Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0011 | 0b101 | TLBI VALE2IS | TLB Invalidate by VA, Last level, EL2, Inner Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0011 | 0b101 | TLBI VALE2ISNXS | TLB Invalidate by VA, Last level, EL2, Inner Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0011 | 0b110 | TLBI VMALLS12E1IS | TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0011 | 0b110 | TLBI VMALLS12E1ISNXS | TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b000 | TLBI IPAS2E1OS | TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b000 | TLBI IPAS2E1OSNXS | TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b001 | TLBI IPAS2E1 | TLB Invalidate by Intermediate Physical Address, Stage 2, EL1 |
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b001 | TLBI IPAS2E1NXS | TLB Invalidate by Intermediate Physical Address, Stage 2, EL1 |
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b010 | TLBI RIPAS2E1 | TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1 |
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b010 | TLBI RIPAS2E1NXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1 |
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b011 | TLBI RIPAS2E1OS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b011 | TLBI RIPAS2E1OSNXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b100 | TLBI IPAS2LE1OS | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b100 | TLBI IPAS2LE1OSNXS | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b101 | TLBI IPAS2LE1 | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1 |

| Register selectors | | | | | Name | Description |
|--------------------|-------|--------|--------|-------|-------------------------------------|--|
| op0 | op1 | CRn | CRm | op2 | | |
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b101 | TLBI IPAS2LE1NXS | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1 |
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b110 | TLBI RIPAS2LE1 | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1 |
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b110 | TLBI RIPAS2LE1NXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1 |
| 0b01 | 0b100 | 0b1000 | 0b0100 | 0b111 | TLBI RIPAS2LE1OS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0100 | 0b111 | TLBI RIPAS2LE1OSNXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0101 | 0b001 | TLBI RVAE2OS | TLB Range Invalidate by VA, EL2, Outer Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0101 | 0b001 | TLBI RVAE2OSNXS | TLB Range Invalidate by VA, EL2, Outer Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0101 | 0b101 | TLBI RVALE2OS | TLB Range Invalidate by VA, Last level, EL2, Outer Shareable |
| 0b01 | 0b100 | 0b1001 | 0b0101 | 0b101 | TLBI RVALE2OSNXS | TLB Range Invalidate by VA, Last level, EL2, Outer Shareable |
| 0b01 | 0b100 | 0b1000 | 0b0110 | 0b001 | TLBI RVAE2 | TLB Range Invalidate by VA, EL2 |
| 0b01 | 0b100 | 0b1001 | 0b0110 | 0b001 | TLBI RVAE2NXS | TLB Range Invalidate by VA, EL2 |
| 0b01 | 0b100 | 0b1000 | 0b0110 | 0b101 | TLBI RVALE2 | TLB Range Invalidate by VA, Last level, EL2 |
| 0b01 | 0b100 | 0b1001 | 0b0110 | 0b101 | TLBI RVALE2NXS | TLB Range Invalidate by VA, Last level, EL2 |
| 0b01 | 0b100 | 0b1000 | 0b0111 | 0b000 | TLBI ALLE2 | TLB Invalidate All, EL2 |
| 0b01 | 0b100 | 0b1001 | 0b0111 | 0b000 | TLBI ALLE2NXS | TLB Invalidate All, EL2 |
| 0b01 | 0b100 | 0b1000 | 0b0111 | 0b001 | TLBI VAE2 | TLB Invalidate by VA, EL2 |
| 0b01 | 0b100 | 0b1001 | 0b0111 | 0b001 | TLBI VAE2NXS | TLB Invalidate by VA, EL2 |
| 0b01 | 0b100 | 0b1000 | 0b0111 | 0b100 | TLBI ALLE1 | TLB Invalidate All, EL1 |
| 0b01 | 0b100 | 0b1001 | 0b0111 | 0b100 | TLBI ALLE1NXS | TLB Invalidate All, EL1 |
| 0b01 | 0b100 | 0b1000 | 0b0111 | 0b101 | TLBI VALE2 | TLB Invalidate by VA, Last level, EL2 |
| 0b01 | 0b100 | 0b1001 | 0b0111 | 0b101 | TLBI VALE2NXS | TLB Invalidate by VA, Last level, EL2 |
| 0b01 | 0b100 | 0b1000 | 0b0111 | 0b110 | TLBI VMALLS12E1 | TLB Invalidate by VMID, All at Stage 1 and 2, EL1 |

| Register selectors | | | | | Name | Description |
|--------------------|-------|--------|--------|-------|------------------------------------|---|
| op0 | op1 | CRn | CRm | op2 | | |
| 0b01 | 0b100 | 0b1001 | 0b0111 | 0b110 | TLBI VMALLS12E1NXS | TLB Invalidate by VMID, All at Stage 1 and 2, EL1 |
| 0b01 | 0b110 | 0b1000 | 0b0001 | 0b000 | TLBI ALLE3OS | TLB Invalidate All, EL3, Outer Shareable |
| 0b01 | 0b110 | 0b1001 | 0b0001 | 0b000 | TLBI ALLE3OSNXS | TLB Invalidate All, EL3, Outer Shareable |
| 0b01 | 0b110 | 0b1000 | 0b0001 | 0b001 | TLBI VAE3OS | TLB Invalidate by VA, EL3, Outer Shareable |
| 0b01 | 0b110 | 0b1001 | 0b0001 | 0b001 | TLBI VAE3OSNXS | TLB Invalidate by VA, EL3, Outer Shareable |
| 0b01 | 0b110 | 0b1000 | 0b0001 | 0b100 | TLBI PAALLOS | TLB Invalidate GPT Information by PA, All Entries, Outer Shareable |
| 0b01 | 0b110 | 0b1000 | 0b0001 | 0b101 | TLBI VALE3OS | TLB Invalidate by VA, Last level, EL3, Outer Shareable |
| 0b01 | 0b110 | 0b1001 | 0b0001 | 0b101 | TLBI VALE3OSNXS | TLB Invalidate by VA, Last level, EL3, Outer Shareable |
| 0b01 | 0b110 | 0b1000 | 0b0010 | 0b001 | TLBI RVAE3IS | TLB Range Invalidate by VA, EL3, Inner Shareable |
| 0b01 | 0b110 | 0b1001 | 0b0010 | 0b001 | TLBI RVAE3ISNXS | TLB Range Invalidate by VA, EL3, Inner Shareable |
| 0b01 | 0b110 | 0b1000 | 0b0010 | 0b101 | TLBI RVALE3IS | TLB Range Invalidate by VA, Last level, EL3, Inner Shareable |
| 0b01 | 0b110 | 0b1001 | 0b0010 | 0b101 | TLBI RVALE3ISNXS | TLB Range Invalidate by VA, Last level, EL3, Inner Shareable |
| 0b01 | 0b110 | 0b1000 | 0b0011 | 0b000 | TLBI ALLE3IS | TLB Invalidate All, EL3, Inner Shareable |
| 0b01 | 0b110 | 0b1001 | 0b0011 | 0b000 | TLBI ALLE3ISNXS | TLB Invalidate All, EL3, Inner Shareable |
| 0b01 | 0b110 | 0b1000 | 0b0011 | 0b001 | TLBI VAE3IS | TLB Invalidate by VA, EL3, Inner Shareable |
| 0b01 | 0b110 | 0b1001 | 0b0011 | 0b001 | TLBI VAE3ISNXS | TLB Invalidate by VA, EL3, Inner Shareable |
| 0b01 | 0b110 | 0b1000 | 0b0011 | 0b101 | TLBI VALE3IS | TLB Invalidate by VA, Last level, EL3, Inner Shareable |
| 0b01 | 0b110 | 0b1001 | 0b0011 | 0b101 | TLBI VALE3ISNXS | TLB Invalidate by VA, Last level, EL3, Inner Shareable |
| 0b01 | 0b110 | 0b1000 | 0b0100 | 0b011 | TLBI RPAOS | TLB Range Invalidate GPT Information by PA, Outer Shareable |
| 0b01 | 0b110 | 0b1000 | 0b0100 | 0b111 | TLBI RPALOS | TLB Range Invalidate GPT Information by PA, Last level, Outer Shareable |
| 0b01 | 0b110 | 0b1000 | 0b0101 | 0b001 | TLBI RVAE3OS | TLB Range Invalidate by VA, EL3, Outer Shareable |
| 0b01 | 0b110 | 0b1001 | 0b0101 | 0b001 | TLBI RVAE3OSNXS | TLB Range Invalidate by VA, EL3, Outer Shareable |

| Register selectors | | | | | Name | Description |
|--------------------|-------|--------|--------|-------|----------------------------------|--|
| op0 | op1 | CRn | CRm | op2 | | |
| 0b01 | 0b110 | 0b1000 | 0b0101 | 0b101 | TLBI RVALE3OS | TLB Range Invalidate by VA, Last level, EL3, Outer Shareable |
| 0b01 | 0b110 | 0b1001 | 0b0101 | 0b101 | TLBI RVALE3OSNXS | TLB Range Invalidate by VA, Last level, EL3, Outer Shareable |
| 0b01 | 0b110 | 0b1000 | 0b0110 | 0b001 | TLBI RVAE3 | TLB Range Invalidate by VA, EL3 |
| 0b01 | 0b110 | 0b1001 | 0b0110 | 0b001 | TLBI RVAE3NXS | TLB Range Invalidate by VA, EL3 |
| 0b01 | 0b110 | 0b1000 | 0b0110 | 0b101 | TLBI RVALE3 | TLB Range Invalidate by VA, Last level, EL3 |
| 0b01 | 0b110 | 0b1001 | 0b0110 | 0b101 | TLBI RVALE3NXS | TLB Range Invalidate by VA, Last level, EL3 |
| 0b01 | 0b110 | 0b1000 | 0b0111 | 0b000 | TLBI ALLE3 | TLB Invalidate All, EL3 |
| 0b01 | 0b110 | 0b1001 | 0b0111 | 0b000 | TLBI ALLE3NXS | TLB Invalidate All, EL3 |
| 0b01 | 0b110 | 0b1000 | 0b0111 | 0b001 | TLBI VAE3 | TLB Invalidate by VA, EL3 |
| 0b01 | 0b110 | 0b1001 | 0b0111 | 0b001 | TLBI VAE3NXS | TLB Invalidate by VA, EL3 |
| 0b01 | 0b110 | 0b1000 | 0b0111 | 0b100 | TLBI PAALL | TLB Invalidate GPT Information by PA, All Entries, Local |
| 0b01 | 0b110 | 0b1000 | 0b0111 | 0b101 | TLBI VALE3 | TLB Invalidate by VA, Last level, EL3 |
| 0b01 | 0b110 | 0b1001 | 0b0111 | 0b101 | TLBI VALE3NXS | TLB Invalidate by VA, Last level, EL3 |

30/09/2021 14:57

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

System Register index by functional group

Below are indexes for registers with the following main functional groups:

- [ID](#)
- [Memory](#)
- [Other](#)
- [Exception](#)
- [Special](#)
- [PSTATE](#)
- [Cache](#)
- [Address](#)
- [TLB](#)
- [PMU](#)
- [Reset](#)
- [Thread](#)
- [IMP DEF](#)
- [Timer](#)
- [Debug](#)
- [CTI](#)
- [Virt](#)
- [Secure](#)
- [Float](#)
- [Legacy](#)
- [Trace](#)
- [GIC](#)
- [GICD](#)
- [GICR](#)
- [GICC](#)
- [GICV](#)
- [GICH](#)
- [GITS](#)
- [RAS](#)
- [MPAM](#)
- [Pointer authentication](#)
- [AMU](#)
- [Root](#)
- [GIC ITS registers](#)

In the ID functional group:

| Exec state | Name | Description |
|------------|--------------------------|--------------------------------------|
| AArch32 | CCSIDR | Current Cache Size ID Register |
| AArch32 | CCSIDR2 | Current Cache Size ID Register 2 |
| AArch32 | CLIDR | Cache Level ID Register |
| AArch32 | CSSELR | Cache Size Selection Register |
| AArch32 | CTR | Cache Type Register |
| AArch32 | ID_AFR0 | Auxiliary Feature Register 0 |
| AArch32 | ID_DFR0 | Debug Feature Register 0 |
| AArch32 | ID_DFR1 | Debug Feature Register 1 |
| AArch32 | ID_ISAR0 | Instruction Set Attribute Register 0 |
| AArch32 | ID_ISAR1 | Instruction Set Attribute Register 1 |
| AArch32 | ID_ISAR2 | Instruction Set Attribute Register 2 |
| AArch32 | ID_ISAR3 | Instruction Set Attribute Register 3 |
| AArch32 | ID_ISAR4 | Instruction Set Attribute Register 4 |
| AArch32 | ID_ISAR5 | Instruction Set Attribute Register 5 |
| AArch32 | ID_ISAR6 | Instruction Set Attribute Register 6 |
| AArch32 | ID_MMFR0 | Memory Model Feature Register 0 |
| AArch32 | ID_MMFR1 | Memory Model Feature Register 1 |
| AArch32 | ID_MMFR2 | Memory Model Feature Register 2 |
| AArch32 | ID_MMFR3 | Memory Model Feature Register 3 |
| AArch32 | ID_MMFR4 | Memory Model Feature Register 4 |
| AArch32 | ID_MMFR5 | Memory Model Feature Register 5 |
| AArch32 | ID_PFR0 | Processor Feature Register 0 |

| Exec state | Name | Description |
|------------|----------------------------------|---|
| AArch32 | ID_PFR1 | Processor Feature Register 1 |
| AArch32 | ID_PFR2 | Processor Feature Register 2 |
| AArch32 | MIDR | Main ID Register |
| AArch32 | MPIDR | Multiprocessor Affinity Register |
| AArch32 | REVIDR | Revision ID Register |
| AArch32 | TCMTR | TCM Type Register |
| AArch32 | TLBTR | TLB Type Register |
| AArch64 | CCSIDR2_EL1 | Current Cache Size ID Register 2 |
| AArch64 | CCSIDR_EL1 | Current Cache Size ID Register |
| AArch64 | CLIDR_EL1 | Cache Level ID Register |
| AArch64 | CSSELR_EL1 | Cache Size Selection Register |
| AArch64 | CTR_EL0 | Cache Type Register |
| AArch64 | DCZID_EL0 | Data Cache Zero ID register |
| AArch64 | GMID_EL1 | Multiple tag transfer ID register |
| AArch64 | ID_AA64AFR0_EL1 | AArch64 Auxiliary Feature Register 0 |
| AArch64 | ID_AA64AFR1_EL1 | AArch64 Auxiliary Feature Register 1 |
| AArch64 | ID_AA64DFR0_EL1 | AArch64 Debug Feature Register 0 |
| AArch64 | ID_AA64DFR1_EL1 | AArch64 Debug Feature Register 1 |
| AArch64 | ID_AA64ISAR0_EL1 | AArch64 Instruction Set Attribute Register 0 |
| AArch64 | ID_AA64ISAR1_EL1 | AArch64 Instruction Set Attribute Register 1 |
| AArch64 | ID_AA64ISAR2_EL1 | AArch64 Instruction Set Attribute Register 2 |
| AArch64 | ID_AA64MMFR0_EL1 | AArch64 Memory Model Feature Register 0 |
| AArch64 | ID_AA64MMFR1_EL1 | AArch64 Memory Model Feature Register 1 |
| AArch64 | ID_AA64MMFR2_EL1 | AArch64 Memory Model Feature Register 2 |
| AArch64 | ID_AA64PFR0_EL1 | AArch64 Processor Feature Register 0 |
| AArch64 | ID_AA64PFR1_EL1 | AArch64 Processor Feature Register 1 |
| AArch64 | ID_AA64SMFR0_EL1 | SME Feature ID register 0 |
| AArch64 | ID_AA64ZFR0_EL1 | SVE Feature ID register 0 |
| AArch64 | ID_AFR0_EL1 | AArch32 Auxiliary Feature Register 0 |
| AArch64 | ID_DFR0_EL1 | AArch32 Debug Feature Register 0 |
| AArch64 | ID_DFR1_EL1 | Debug Feature Register 1 |
| AArch64 | ID_ISAR0_EL1 | AArch32 Instruction Set Attribute Register 0 |
| AArch64 | ID_ISAR1_EL1 | AArch32 Instruction Set Attribute Register 1 |
| AArch64 | ID_ISAR2_EL1 | AArch32 Instruction Set Attribute Register 2 |
| AArch64 | ID_ISAR3_EL1 | AArch32 Instruction Set Attribute Register 3 |
| AArch64 | ID_ISAR4_EL1 | AArch32 Instruction Set Attribute Register 4 |
| AArch64 | ID_ISAR5_EL1 | AArch32 Instruction Set Attribute Register 5 |
| AArch64 | ID_ISAR6_EL1 | AArch32 Instruction Set Attribute Register 6 |
| AArch64 | ID_MMFR0_EL1 | AArch32 Memory Model Feature Register 0 |
| AArch64 | ID_MMFR1_EL1 | AArch32 Memory Model Feature Register 1 |
| AArch64 | ID_MMFR2_EL1 | AArch32 Memory Model Feature Register 2 |
| AArch64 | ID_MMFR3_EL1 | AArch32 Memory Model Feature Register 3 |
| AArch64 | ID_MMFR4_EL1 | AArch32 Memory Model Feature Register 4 |
| AArch64 | ID_MMFR5_EL1 | AArch32 Memory Model Feature Register 5 |
| AArch64 | ID_PFR0_EL1 | AArch32 Processor Feature Register 0 |
| AArch64 | ID_PFR1_EL1 | AArch32 Processor Feature Register 1 |
| AArch64 | ID_PFR2_EL1 | AArch32 Processor Feature Register 2 |
| AArch64 | MIDR_EL1 | Main ID Register |
| AArch64 | MPAMIDR_EL1 | MPAM ID Register (EL1) |
| AArch64 | MPIDR_EL1 | Multiprocessor Affinity Register |
| AArch64 | REVIDR_EL1 | Revision ID Register |
| AArch64 | SMIDR_EL1 | Streaming Mode Identification Register |
| External | EDAA32PFR | External Debug Auxiliary Processor Feature Register |
| External | EDDFR | External Debug Feature Register |
| External | EDPFR | External Debug Processor Feature Register |
| External | MIDR_EL1 | Main ID Register |

In the Memory functional group:

| Exec state | Name | Description |
|------------|----------------------------|---|
| AArch32 | AMAIRO | Auxiliary Memory Attribute Indirection Register 0 |
| AArch32 | AMAIR1 | Auxiliary Memory Attribute Indirection Register 1 |
| AArch32 | CONTEXTIDR | Context ID Register |

| Exec state | Name | Description |
|------------|--------------------------------|---|
| AArch32 | DACR | Domain Access Control Register |
| AArch32 | HAMAIR0 | Hyp Auxiliary Memory Attribute Indirection Register 0 |
| AArch32 | HAMAIR1 | Hyp Auxiliary Memory Attribute Indirection Register 1 |
| AArch32 | HMAIR0 | Hyp Memory Attribute Indirection Register 0 |
| AArch32 | HMAIR1 | Hyp Memory Attribute Indirection Register 1 |
| AArch32 | HTCR | Hyp Translation Control Register |
| AArch32 | HTTBR | Hyp Translation Table Base Register |
| AArch32 | MAIR0 | Memory Attribute Indirection Register 0 |
| AArch32 | MAIR1 | Memory Attribute Indirection Register 1 |
| AArch32 | NMRR | Normal Memory Remap Register |
| AArch32 | PRRR | Primary Region Remap Register |
| AArch32 | TTBCR | Translation Table Base Control Register |
| AArch32 | TTBCR2 | Translation Table Base Control Register 2 |
| AArch32 | TTBR0 | Translation Table Base Register 0 |
| AArch32 | TTBR1 | Translation Table Base Register 1 |
| AArch32 | VTCR | Virtualization Translation Control Register |
| AArch32 | VTTBR | Virtualization Translation Table Base Register |
| AArch64 | AMAIR_EL1 | Auxiliary Memory Attribute Indirection Register (EL1) |
| AArch64 | AMAIR_EL2 | Auxiliary Memory Attribute Indirection Register (EL2) |
| AArch64 | AMAIR_EL3 | Auxiliary Memory Attribute Indirection Register (EL3) |
| AArch64 | CONTEXTIDR_EL1 | Context ID Register (EL1) |
| AArch64 | CONTEXTIDR_EL2 | Context ID Register (EL2) |
| AArch64 | DACR32_EL2 | Domain Access Control Register |
| AArch64 | GPCCR_EL3 | Granule Protection Check Control Register (EL3) |
| AArch64 | GPTBR_EL3 | Granule Protection Table Base Register |
| AArch64 | LORC_EL1 | LORegion Control (EL1) |
| AArch64 | LOREA_EL1 | LORegion End Address (EL1) |
| AArch64 | LORID_EL1 | LORegionID (EL1) |
| AArch64 | LORN_EL1 | LORegion Number (EL1) |
| AArch64 | LORSA_EL1 | LORegion Start Address (EL1) |
| AArch64 | MAIR_EL1 | Memory Attribute Indirection Register (EL1) |
| AArch64 | MAIR_EL2 | Memory Attribute Indirection Register (EL2) |
| AArch64 | MAIR_EL3 | Memory Attribute Indirection Register (EL3) |
| AArch64 | TCR_EL1 | Translation Control Register (EL1) |
| AArch64 | TCR_EL2 | Translation Control Register (EL2) |
| AArch64 | TCR_EL3 | Translation Control Register (EL3) |
| AArch64 | TTBR0_EL1 | Translation Table Base Register 0 (EL1) |
| AArch64 | TTBR0_EL2 | Translation Table Base Register 0 (EL2) |
| AArch64 | TTBR0_EL3 | Translation Table Base Register 0 (EL3) |
| AArch64 | TTBR1_EL1 | Translation Table Base Register 1 (EL1) |
| AArch64 | TTBR1_EL2 | Translation Table Base Register 1 (EL2) |
| AArch64 | VTCR_EL2 | Virtualization Translation Control Register |
| AArch64 | VTTBR_EL2 | Virtualization Translation Table Base Register |

In the Other functional group:

| Exec state | Name | Description |
|------------|------------------------------|---|
| AArch32 | CPACR | Architectural Feature Access Control Register |
| AArch32 | SCTLR | System Control Register |
| AArch64 | CPACR_EL1 | Architectural Feature Access Control Register |
| AArch64 | SCTLR_EL1 | System Control Register (EL1) |
| AArch64 | SCTLR_EL3 | System Control Register (EL3) |
| AArch64 | SMCR_EL1 | SME Control Register (EL1) |
| AArch64 | SMCR_EL2 | SME Control Register (EL2) |
| AArch64 | SMCR_EL3 | SME Control Register (EL3) |
| AArch64 | SMPRIMAP_EL2 | Streaming Mode Priority Mapping Register |
| AArch64 | SMPRI_EL1 | Streaming Mode Priority Register |
| AArch64 | ZCR_EL1 | SVE Control Register (EL1) |
| AArch64 | ZCR_EL2 | SVE Control Register (EL2) |
| AArch64 | ZCR_EL3 | SVE Control Register (EL3) |

In the Exception functional group:

| Exec state | Name | Description |
|------------|----------------------------|---|
| AArch32 | ADFSR | Auxiliary Data Fault Status Register |
| AArch32 | AIFSR | Auxiliary Instruction Fault Status Register |
| AArch32 | DFAR | Data Fault Address Register |
| AArch32 | DFSR | Data Fault Status Register |
| AArch32 | HADFSR | Hyp Auxiliary Data Fault Status Register |
| AArch32 | HAIFSR | Hyp Auxiliary Instruction Fault Status Register |
| AArch32 | HDFAR | Hyp Data Fault Address Register |
| AArch32 | HIFAR | Hyp Instruction Fault Address Register |
| AArch32 | HPFAR | Hyp IPA Fault Address Register |
| AArch32 | HSR | Hyp Syndrome Register |
| AArch32 | HVBAR | Hyp Vector Base Address Register |
| AArch32 | IFAR | Instruction Fault Address Register |
| AArch32 | IFSR | Instruction Fault Status Register |
| AArch32 | ISR | Interrupt Status Register |
| AArch32 | MVBAR | Monitor Vector Base Address Register |
| AArch32 | VBAR | Vector Base Address Register |
| AArch64 | AFSR0_EL1 | Auxiliary Fault Status Register 0 (EL1) |
| AArch64 | AFSR0_EL2 | Auxiliary Fault Status Register 0 (EL2) |
| AArch64 | AFSR0_EL3 | Auxiliary Fault Status Register 0 (EL3) |
| AArch64 | AFSR1_EL1 | Auxiliary Fault Status Register 1 (EL1) |
| AArch64 | AFSR1_EL2 | Auxiliary Fault Status Register 1 (EL2) |
| AArch64 | AFSR1_EL3 | Auxiliary Fault Status Register 1 (EL3) |
| AArch64 | ESR_EL1 | Exception Syndrome Register (EL1) |
| AArch64 | ESR_EL2 | Exception Syndrome Register (EL2) |
| AArch64 | ESR_EL3 | Exception Syndrome Register (EL3) |
| AArch64 | FAR_EL1 | Fault Address Register (EL1) |
| AArch64 | FAR_EL2 | Fault Address Register (EL2) |
| AArch64 | FAR_EL3 | Fault Address Register (EL3) |
| AArch64 | HPFAR_EL2 | Hypervisor IPA Fault Address Register |
| AArch64 | IFSR32_EL2 | Instruction Fault Status Register (EL2) |
| AArch64 | ISR_EL1 | Interrupt Status Register |
| AArch64 | MFAR_EL3 | PA Fault Address Register |
| AArch64 | VBAR_EL1 | Vector Base Address Register (EL1) |
| AArch64 | VBAR_EL2 | Vector Base Address Register (EL2) |
| AArch64 | VBAR_EL3 | Vector Base Address Register (EL3) |

In the Special functional group:

| Exec state | Name | Description |
|------------|--------------------------|---|
| AArch32 | DLR | Debug Link Register |
| AArch32 | DSPSR | Debug Saved Program Status Register |
| AArch32 | ELR_hyp | Exception Link Register (Hyp mode) |
| AArch32 | SPSR | Saved Program Status Register |
| AArch32 | SPSR_abt | Saved Program Status Register (Abort mode) |
| AArch32 | SPSR_fiq | Saved Program Status Register (FIQ mode) |
| AArch32 | SPSR_hyp | Saved Program Status Register (Hyp mode) |
| AArch32 | SPSR_irq | Saved Program Status Register (IRQ mode) |
| AArch32 | SPSR_mon | Saved Program Status Register (Monitor mode) |
| AArch32 | SPSR_svc | Saved Program Status Register (Supervisor mode) |
| AArch32 | SPSR_und | Saved Program Status Register (Undefined mode) |
| AArch64 | ELR_EL1 | Exception Link Register (EL1) |
| AArch64 | ELR_EL2 | Exception Link Register (EL2) |
| AArch64 | ELR_EL3 | Exception Link Register (EL3) |
| AArch64 | SPSR_EL1 | Saved Program Status Register (EL1) |
| AArch64 | SPSR_EL2 | Saved Program Status Register (EL2) |
| AArch64 | SPSR_EL3 | Saved Program Status Register (EL3) |
| AArch64 | SPSR_abt | Saved Program Status Register (Abort mode) |
| AArch64 | SPSR_fiq | Saved Program Status Register (FIQ mode) |
| AArch64 | SPSR_irq | Saved Program Status Register (IRQ mode) |
| AArch64 | SPSR_und | Saved Program Status Register (Undefined mode) |
| AArch64 | SP_ELO | Stack Pointer (EL0) |

| Exec state | Name | Description |
|------------|------------------------|---------------------|
| AArch64 | SP_EL1 | Stack Pointer (EL1) |
| AArch64 | SP_EL2 | Stack Pointer (EL2) |
| AArch64 | SP_EL3 | Stack Pointer (EL3) |

In the PSTATE functional group:

| Exec state | Name | Description |
|------------|---------------------------|-------------------------------------|
| AArch32 | APSR | Application Program Status Register |
| AArch32 | CPSR | Current Program Status Register |
| AArch64 | ALLINT | All Interrupt Mask Bit |
| AArch64 | CurrentEL | Current Exception Level |
| AArch64 | DAIF | Interrupt Mask Bits |
| AArch64 | DIT | Data Independent Timing |
| AArch64 | NZCV | Condition Flags |
| AArch64 | PAN | Privileged Access Never |
| AArch64 | SPSel | Stack Pointer Select |
| AArch64 | SSBS | Speculative Store Bypass Safe |
| AArch64 | SVCR | Streaming Vector Control Register |
| AArch64 | TCO | Tag Check Override |
| AArch64 | UAO | User Access Override |

In the Cache functional group:

| Exec state | Name | Description |
|------------|-----------------------------|--|
| AArch32 | BPIALL | Branch Predictor Invalidate All |
| AArch32 | BPIALLIS | Branch Predictor Invalidate All, Inner Shareable |
| AArch32 | BPIMVA | Branch Predictor Invalidate by VA |
| AArch32 | DCCIMVAC | Data Cache line Clean and Invalidate by VA to PoC |
| AArch32 | DCCISW | Data Cache line Clean and Invalidate by Set/Way |
| AArch32 | DCCMVAC | Data Cache line Clean by VA to PoC |
| AArch32 | DCCMVAU | Data Cache line Clean by VA to PoU |
| AArch32 | DCCSW | Data Cache line Clean by Set/Way |
| AArch32 | DCIMVAC | Data Cache line Invalidate by VA to PoC |
| AArch32 | DCISW | Data Cache line Invalidate by Set/Way |
| AArch32 | ICIALLU | Instruction Cache Invalidate All to PoU |
| AArch32 | ICIALLUIS | Instruction Cache Invalidate All to PoU, Inner Shareable |
| AArch32 | ICIMVAU | Instruction Cache line Invalidate by VA to PoU |
| AArch64 | DC CGDSW | Clean of Data and Allocation Tags by Set/Way |
| AArch64 | DC CGDVAC | Clean of Data and Allocation Tags by VA to PoC |
| AArch64 | DC CGDVADP | Clean of Data and Allocation Tags by VA to PoDP |
| AArch64 | DC CGDVAP | Clean of Data and Allocation Tags by VA to PoP |
| AArch64 | DC CGSW | Clean of Allocation Tags by Set/Way |
| AArch64 | DC CGVAC | Clean of Allocation Tags by VA to PoC |
| AArch64 | DC CGVADP | Clean of Allocation Tags by VA to PoDP |
| AArch64 | DC CGVAP | Clean of Allocation Tags by VA to PoP |
| AArch64 | DC CIGDPAPA | Clean and Invalidate of Data and Allocation Tags by PA to PoPA |
| AArch64 | DC CIGDSW | Clean and Invalidate of Data and Allocation Tags by Set/Way |
| AArch64 | DC CIGDVAC | Clean and Invalidate of Data and Allocation Tags by VA to PoC |
| AArch64 | DC CIGSW | Clean and Invalidate of Allocation Tags by Set/Way |
| AArch64 | DC CIGVAC | Clean and Invalidate of Allocation Tags by VA to PoC |
| AArch64 | DC CIPAPA | Data or unified Cache line Clean and Invalidate by PA to PoPA |
| AArch64 | DC CISW | Data or unified Cache line Clean and Invalidate by Set/Way |
| AArch64 | DC CIVAC | Data or unified Cache line Clean and Invalidate by VA to PoC |
| AArch64 | DC CSW | Data or unified Cache line Clean by Set/Way |
| AArch64 | DC CVAC | Data or unified Cache line Clean by VA to PoC |
| AArch64 | DC CVADP | Data or unified Cache line Clean by VA to PoDP |
| AArch64 | DC CVAP | Data or unified Cache line Clean by VA to PoP |
| AArch64 | DC CVAU | Data or unified Cache line Clean by VA to PoU |
| AArch64 | DC GVA | Data Cache set Allocation Tag by VA |
| AArch64 | DC GZVA | Data Cache set Allocation Tags and Zero by VA |
| AArch64 | DC IGDSW | Invalidate of Data and Allocation Tags by Set/Way |
| AArch64 | DC IGDVAC | Invalidate of Data and Allocation Tags by VA to PoC |

| Exec state | Name | Description |
|------------|----------------------------|--|
| AArch64 | DC IGSW | Invalidate of Allocation Tags by Set/Way |
| AArch64 | DC IGVAC | Invalidate of Allocation Tags by VA to PoC |
| AArch64 | DC ISW | Data or unified Cache line Invalidate by Set/Way |
| AArch64 | DC IVAC | Data or unified Cache line Invalidate by VA to PoC |
| AArch64 | DC ZVA | Data Cache Zero by VA |
| AArch64 | IC IALLU | Instruction Cache Invalidate All to PoU |
| AArch64 | IC IALLUIS | Instruction Cache Invalidate All to PoU, Inner Shareable |
| AArch64 | IC IVAU | Instruction Cache line Invalidate by VA to PoU |

In the Address functional group:

| Exec state | Name | Description |
|------------|----------------------------|---|
| AArch32 | ATS12NSOPR | Address Translate Stages 1 and 2 Non-secure Only PL1 Read |
| AArch32 | ATS12NSOPW | Address Translate Stages 1 and 2 Non-secure Only PL1 Write |
| AArch32 | ATS12NSOUR | Address Translate Stages 1 and 2 Non-secure Only Unprivileged Read |
| AArch32 | ATS12NSOUW | Address Translate Stages 1 and 2 Non-secure Only Unprivileged Write |
| AArch32 | ATS1CPR | Address Translate Stage 1 Current state PL1 Read |
| AArch32 | ATS1CPRP | Address Translate Stage 1 Current state PL1 Read PAN |
| AArch32 | ATS1CPW | Address Translate Stage 1 Current state PL1 Write |
| AArch32 | ATS1CPWP | Address Translate Stage 1 Current state PL1 Write PAN |
| AArch32 | ATS1CUR | Address Translate Stage 1 Current state Unprivileged Read |
| AArch32 | ATS1CUW | Address Translate Stage 1 Current state Unprivileged Write |
| AArch32 | ATS1HR | Address Translate Stage 1 Hyp mode Read |
| AArch32 | ATS1HW | Address Translate Stage 1 Hyp mode Write |
| AArch32 | PAR | Physical Address Register |
| AArch64 | AT S12E0R | Address Translate Stages 1 and 2 EL0 Read |
| AArch64 | AT S12E0W | Address Translate Stages 1 and 2 EL0 Write |
| AArch64 | AT S12E1R | Address Translate Stages 1 and 2 EL1 Read |
| AArch64 | AT S12E1W | Address Translate Stages 1 and 2 EL1 Write |
| AArch64 | AT S1E0R | Address Translate Stage 1 EL0 Read |
| AArch64 | AT S1E0W | Address Translate Stage 1 EL0 Write |
| AArch64 | AT S1E1R | Address Translate Stage 1 EL1 Read |
| AArch64 | AT S1E1RP | Address Translate Stage 1 EL1 Read PAN |
| AArch64 | AT S1E1W | Address Translate Stage 1 EL1 Write |
| AArch64 | AT S1E1WP | Address Translate Stage 1 EL1 Write PAN |
| AArch64 | AT S1E2R | Address Translate Stage 1 EL2 Read |
| AArch64 | AT S1E2W | Address Translate Stage 1 EL2 Write |
| AArch64 | AT S1E3R | Address Translate Stage 1 EL3 Read |
| AArch64 | AT S1E3W | Address Translate Stage 1 EL3 Write |
| AArch64 | PAR_EL1 | Physical Address Register |

In the TLB functional group:

| Exec state | Name | Description |
|------------|-------------------------------|---|
| AArch32 | CFPRCTX | Control Flow Prediction Restriction by Context |
| AArch32 | CPPRCTX | Cache Prefetch Prediction Restriction by Context |
| AArch32 | DTLBIALL | Data TLB Invalidate All |
| AArch32 | DTLBIASID | Data TLB Invalidate by ASID match |
| AArch32 | DTLBIMVA | Data TLB Invalidate by VA |
| AArch32 | DVPRCTX | Data Value Prediction Restriction by Context |
| AArch32 | ITLBIALL | Instruction TLB Invalidate All |
| AArch32 | ITLBIASID | Instruction TLB Invalidate by ASID match |
| AArch32 | ITLBIMVA | Instruction TLB Invalidate by VA |
| AArch32 | TLBIALL | TLB Invalidate All |
| AArch32 | TLBIALLH | TLB Invalidate All, Hyp mode |
| AArch32 | TLBIALLHIS | TLB Invalidate All, Hyp mode, Inner Shareable |
| AArch32 | TLBIALLIS | TLB Invalidate All, Inner Shareable |
| AArch32 | TLBIALLNSNH | TLB Invalidate All, Non-Secure Non-Hyp |
| AArch32 | TLBIALLNSNHIS | TLB Invalidate All, Non-Secure Non-Hyp, Inner Shareable |
| AArch32 | TLBIASID | TLB Invalidate by ASID match |
| AArch32 | TLBIASIDIS | TLB Invalidate by ASID match, Inner Shareable |

| Exec state | Name | Description |
|------------|---|--|
| AArch32 | TLBIIPAS2 | TLB Invalidate by Intermediate Physical Address, Stage 2 |
| AArch32 | TLBIIPAS2IS | TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable |
| AArch32 | TLBIIPAS2L | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level |
| AArch32 | TLBIIPAS2LIS | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable |
| AArch32 | TLBIMVA | TLB Invalidate by VA |
| AArch32 | TLBIMVAA | TLB Invalidate by VA, All ASID |
| AArch32 | TLBIMVAAIS | TLB Invalidate by VA, All ASID, Inner Shareable |
| AArch32 | TLBIMVAAL | TLB Invalidate by VA, All ASID, Last level |
| AArch32 | TLBIMVAALIS | TLB Invalidate by VA, All ASID, Last level, Inner Shareable |
| AArch32 | TLBIMVAH | TLB Invalidate by VA, Hyp mode |
| AArch32 | TLBIMVAHIS | TLB Invalidate by VA, Hyp mode, Inner Shareable |
| AArch32 | TLBIMVAIS | TLB Invalidate by VA, Inner Shareable |
| AArch32 | TLBIMVAL | TLB Invalidate by VA, Last level |
| AArch32 | TLBIMVALH | TLB Invalidate by VA, Last level, Hyp mode |
| AArch32 | TLBIMVALHIS | TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable |
| AArch32 | TLBIMVALIS | TLB Invalidate by VA, Last level, Inner Shareable |
| AArch64 | TLBI ALLE1, TLBI ALLE1NXS | TLB Invalidate All, EL1 |
| AArch64 | TLBI ALLE1IS, TLBI ALLE1ISNXS | TLB Invalidate All, EL1, Inner Shareable |
| AArch64 | TLBI ALLE1OS, TLBI ALLE1OSNXS | TLB Invalidate All, EL1, Outer Shareable |
| AArch64 | TLBI ALLE2, TLBI ALLE2NXS | TLB Invalidate All, EL2 |
| AArch64 | TLBI ALLE2IS, TLBI ALLE2ISNXS | TLB Invalidate All, EL2, Inner Shareable |
| AArch64 | TLBI ALLE2OS, TLBI ALLE2OSNXS | TLB Invalidate All, EL2, Outer Shareable |
| AArch64 | TLBI ALLE3, TLBI ALLE3NXS | TLB Invalidate All, EL3 |
| AArch64 | TLBI ALLE3IS, TLBI ALLE3ISNXS | TLB Invalidate All, EL3, Inner Shareable |
| AArch64 | TLBI ALLE3OS, TLBI ALLE3OSNXS | TLB Invalidate All, EL3, Outer Shareable |
| AArch64 | TLBI ASIDE1, TLBI ASIDE1NXS | TLB Invalidate by ASID, EL1 |
| AArch64 | TLBI ASIDE1IS, TLBI ASIDE1ISNXS | TLB Invalidate by ASID, EL1, Inner Shareable |
| AArch64 | TLBI ASIDE1OS, TLBI ASIDE1OSNXS | TLB Invalidate by ASID, EL1, Outer Shareable |
| AArch64 | TLBI IPAS2E1, TLBI IPAS2E1NXS | TLB Invalidate by Intermediate Physical Address, Stage 2, EL1 |
| AArch64 | TLBI IPAS2E1IS, TLBI IPAS2E1ISNXS | TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable |
| AArch64 | TLBI IPAS2E1OS, TLBI IPAS2E1OSNXS | TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable |
| AArch64 | TLBI IPAS2LE1, TLBI IPAS2LE1NXS | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1 |
| AArch64 | TLBI IPAS2LE1IS, TLBI IPAS2LE1ISNXS | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable |
| AArch64 | TLBI IPAS2LE1OS, TLBI IPAS2LE1OSNXS | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable |
| AArch64 | TLBI PAALL | TLB Invalidate GPT Information by PA, All Entries, Local |
| AArch64 | TLBI PAALLOS | TLB Invalidate GPT Information by PA, All Entries, Outer Shareable |
| AArch64 | TLBI RIPAS2E1, TLBI RIPAS2E1NXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1 |
| AArch64 | TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable |
| AArch64 | TLBI RIPAS2E1OS, TLBI RIPAS2E1OSNXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable |
| AArch64 | TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1 |
| AArch64 | TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable |

| Exec state | Name | Description |
|------------|---|--|
| AArch64 | TLBI RIPS2LE1OS, TLBI RIPS2LE1OSNXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable |
| AArch64 | TLBI RPALOS | TLB Range Invalidate GPT Information by PA, Last level, Outer Shareable |
| AArch64 | TLBI RPAOS | TLB Range Invalidate GPT Information by PA, Outer Shareable |
| AArch64 | TLBI RVAAE1, TLBI RVAAE1NXS | TLB Range Invalidate by VA, All ASID, EL1 |
| AArch64 | TLBI RVAAE1IS, TLBI RVAAE1ISNXS | TLB Range Invalidate by VA, All ASID, EL1, Inner Shareable |
| AArch64 | TLBI RVAAE1OS, TLBI RVAAE1OSNXS | TLB Range Invalidate by VA, All ASID, EL1, Outer Shareable |
| AArch64 | TLBI RVAAL1, TLBI RVAAL1NXS | TLB Range Invalidate by VA, All ASID, Last level, EL1 |
| AArch64 | TLBI RVAAL1IS, TLBI RVAAL1ISNXS | TLB Range Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable |
| AArch64 | TLBI RVAAL1OS, TLBI RVAAL1OSNXS | TLB Range Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable |
| AArch64 | TLBI RVAE1, TLBI RVAE1NXS | TLB Range Invalidate by VA, EL1 |
| AArch64 | TLBI RVAE1IS, TLBI RVAE1ISNXS | TLB Range Invalidate by VA, EL1, Inner Shareable |
| AArch64 | TLBI RVAE1OS, TLBI RVAE1OSNXS | TLB Range Invalidate by VA, EL1, Outer Shareable |
| AArch64 | TLBI RVAE2, TLBI RVAE2NXS | TLB Range Invalidate by VA, EL2 |
| AArch64 | TLBI RVAE2IS, TLBI RVAE2ISNXS | TLB Range Invalidate by VA, EL2, Inner Shareable |
| AArch64 | TLBI RVAE2OS, TLBI RVAE2OSNXS | TLB Range Invalidate by VA, EL2, Outer Shareable |
| AArch64 | TLBI RVAE3, TLBI RVAE3NXS | TLB Range Invalidate by VA, EL3 |
| AArch64 | TLBI RVAE3IS, TLBI RVAE3ISNXS | TLB Range Invalidate by VA, EL3, Inner Shareable |
| AArch64 | TLBI RVAE3OS, TLBI RVAE3OSNXS | TLB Range Invalidate by VA, EL3, Outer Shareable |
| AArch64 | TLBI RVALE1, TLBI RVALE1NXS | TLB Range Invalidate by VA, Last level, EL1 |
| AArch64 | TLBI RVALE1IS, TLBI RVALE1ISNXS | TLB Range Invalidate by VA, Last level, EL1, Inner Shareable |
| AArch64 | TLBI RVALE1OS, TLBI RVALE1OSNXS | TLB Range Invalidate by VA, Last level, EL1, Outer Shareable |
| AArch64 | TLBI RVALE2, TLBI RVALE2NXS | TLB Range Invalidate by VA, Last level, EL2 |
| AArch64 | TLBI RVALE2IS, TLBI RVALE2ISNXS | TLB Range Invalidate by VA, Last level, EL2, Inner Shareable |
| AArch64 | TLBI RVALE2OS, TLBI RVALE2OSNXS | TLB Range Invalidate by VA, Last level, EL2, Outer Shareable |
| AArch64 | TLBI RVALE3, TLBI RVALE3NXS | TLB Range Invalidate by VA, Last level, EL3 |
| AArch64 | TLBI RVALE3IS, TLBI RVALE3ISNXS | TLB Range Invalidate by VA, Last level, EL3, Inner Shareable |
| AArch64 | TLBI RVALE3OS, TLBI RVALE3OSNXS | TLB Range Invalidate by VA, Last level, EL3, Outer Shareable |
| AArch64 | TLBI VAAE1, TLBI VAAE1NXS | TLB Invalidate by VA, All ASID, EL1 |
| AArch64 | TLBI VAAE1IS, TLBI VAAE1ISNXS | TLB Invalidate by VA, All ASID, EL1, Inner Shareable |
| AArch64 | TLBI VAAE1OS, TLBI VAAE1OSNXS | TLB Invalidate by VA, All ASID, EL1, Outer Shareable |
| AArch64 | TLBI VAALE1, TLBI VAALE1NXS | TLB Invalidate by VA, All ASID, Last level, EL1 |
| AArch64 | TLBI VAALE1IS, TLBI VAALE1ISNXS | TLB Invalidate by VA, All ASID, Last Level, EL1, Inner Shareable |
| AArch64 | TLBI VAALE1OS, TLBI VAALE1OSNXS | TLB Invalidate by VA, All ASID, Last Level, EL1, Outer Shareable |
| AArch64 | TLBI VAE1, TLBI VAE1NXS | TLB Invalidate by VA, EL1 |
| AArch64 | TLBI VAE1IS, TLBI VAE1ISNXS | TLB Invalidate by VA, EL1, Inner Shareable |
| AArch64 | TLBI VAE1OS, TLBI VAE1OSNXS | TLB Invalidate by VA, EL1, Outer Shareable |
| AArch64 | TLBI VAE2, TLBI VAE2NXS | TLB Invalidate by VA, EL2 |
| AArch64 | TLBI VAE2IS, TLBI VAE2ISNXS | TLB Invalidate by VA, EL2, Inner Shareable |
| AArch64 | TLBI VAE2OS, TLBI VAE2OSNXS | TLB Invalidate by VA, EL2, Outer Shareable |
| AArch64 | TLBI VAE3, TLBI VAE3NXS | TLB Invalidate by VA, EL3 |
| AArch64 | TLBI VAE3IS, TLBI VAE3ISNXS | TLB Invalidate by VA, EL3, Inner Shareable |
| AArch64 | TLBI VAE3OS, TLBI VAE3OSNXS | TLB Invalidate by VA, EL3, Outer Shareable |

| Exec state | Name | Description |
|------------|---|--|
| AArch64 | TLBI VALE1, TLBI VALE1NXS | TLB Invalidate by VA, Last level, EL1 |
| AArch64 | TLBI VALE1IS, TLBI VALE1ISNXS | TLB Invalidate by VA, Last level, EL1, Inner Shareable |
| AArch64 | TLBI VALE1OS, TLBI VALE1OSNXS | TLB Invalidate by VA, Last level, EL1, Outer Shareable |
| AArch64 | TLBI VALE2, TLBI VALE2NXS | TLB Invalidate by VA, Last level, EL2 |
| AArch64 | TLBI VALE2IS, TLBI VALE2ISNXS | TLB Invalidate by VA, Last level, EL2, Inner Shareable |
| AArch64 | TLBI VALE2OS, TLBI VALE2OSNXS | TLB Invalidate by VA, Last level, EL2, Outer Shareable |
| AArch64 | TLBI VALE3, TLBI VALE3NXS | TLB Invalidate by VA, Last level, EL3 |
| AArch64 | TLBI VALE3IS, TLBI VALE3ISNXS | TLB Invalidate by VA, Last level, EL3, Inner Shareable |
| AArch64 | TLBI VALE3OS, TLBI VALE3OSNXS | TLB Invalidate by VA, Last level, EL3, Outer Shareable |
| AArch64 | TLBI VMALLE1, TLBI VMALLE1NXS | TLB Invalidate by VMID, All at stage 1, EL1 |
| AArch64 | TLBI VMALLE1IS, TLBI VMALLE1ISNXS | TLB Invalidate by VMID, All at stage 1, EL1, Inner Shareable |
| AArch64 | TLBI VMALLE1OS, TLBI VMALLE1OSNXS | TLB Invalidate by VMID, All at stage 1, EL1, Outer Shareable |
| AArch64 | TLBI VMALLS12E1, TLBI VMALLS12E1NXS | TLB Invalidate by VMID, All at Stage 1 and 2, EL1 |
| AArch64 | TLBI VMALLS12E1IS, TLBI VMALLS12E1ISNXS | TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Inner Shareable |
| AArch64 | TLBI VMALLS12E1OS, TLBI VMALLS12E1OSNXS | TLB Invalidate by VMID, All at Stage 1 and 2, EL1, Outer Shareable |

In the PMU functional group:

| Exec state | Name | Description |
|------------|--|---|
| AArch32 | PMCCFILTR | Performance Monitors Cycle Count Filter Register |
| AArch32 | PMCCNTR | Performance Monitors Cycle Count Register |
| AArch32 | PMCEID0 | Performance Monitors Common Event Identification register 0 |
| AArch32 | PMCEID1 | Performance Monitors Common Event Identification register 1 |
| AArch32 | PMCEID2 | Performance Monitors Common Event Identification register 2 |
| AArch32 | PMCEID3 | Performance Monitors Common Event Identification register 3 |
| AArch32 | PMCNTENCLR | Performance Monitors Count Enable Clear register |
| AArch32 | PMCNTENSET | Performance Monitors Count Enable Set register |
| AArch32 | PMCR | Performance Monitors Control Register |
| AArch32 | PMEVCNTR<n> | Performance Monitors Event Count Registers |
| AArch32 | PMEVTYPER<n> | Performance Monitors Event Type Registers |
| AArch32 | PMINTENCLR | Performance Monitors Interrupt Enable Clear register |
| AArch32 | PMINTENSET | Performance Monitors Interrupt Enable Set register |
| AArch32 | PMMIR | Performance Monitors Machine Identification Register |
| AArch32 | PMOVSr | Performance Monitors Overflow Flag Status Register |
| AArch32 | PMOVSSET | Performance Monitors Overflow Flag Status Set register |
| AArch32 | PMSELR | Performance Monitors Event Counter Selection Register |
| AArch32 | PMSWINC | Performance Monitors Software Increment register |
| AArch32 | PMUSERENR | Performance Monitors User Enable Register |
| AArch32 | PMXEVCNTR | Performance Monitors Selected Event Count Register |
| AArch32 | PMXEVTYPER | Performance Monitors Selected Event Type Register |
| AArch64 | PMCCFILTR_EL0 | Performance Monitors Cycle Count Filter Register |
| AArch64 | PMCCNTR_EL0 | Performance Monitors Cycle Count Register |
| AArch64 | PMCEID0_EL0 | Performance Monitors Common Event Identification register 0 |
| AArch64 | PMCEID1_EL0 | Performance Monitors Common Event Identification register 1 |
| AArch64 | PMCNTENCLR_EL0 | Performance Monitors Count Enable Clear register |
| AArch64 | PMCNTENSET_EL0 | Performance Monitors Count Enable Set register |
| AArch64 | PMCR_EL0 | Performance Monitors Control Register |
| AArch64 | PMEVCNTR<n>_EL0 | Performance Monitors Event Count Registers |
| AArch64 | PMEVTYPER<n>_EL0 | Performance Monitors Event Type Registers |
| AArch64 | PMINTENCLR_EL1 | Performance Monitors Interrupt Enable Clear register |
| AArch64 | PMINTENSET_EL1 | Performance Monitors Interrupt Enable Set register |

| Exec state | Name | Description |
|------------|--|---|
| AArch64 | PMMIR_EL1 | Performance Monitors Machine Identification Register |
| AArch64 | PMOVSCLR_EL0 | Performance Monitors Overflow Flag Status Clear Register |
| AArch64 | PMOVSSSET_EL0 | Performance Monitors Overflow Flag Status Set register |
| AArch64 | PMSELR_EL0 | Performance Monitors Event Counter Selection Register |
| AArch64 | PMSWINC_EL0 | Performance Monitors Software Increment register |
| AArch64 | PMUSERENR_EL0 | Performance Monitors User Enable Register |
| AArch64 | PMXVCNTR_EL0 | Performance Monitors Selected Event Count Register |
| AArch64 | PMXEVTYPER_EL0 | Performance Monitors Selected Event Type Register |
| External | PMAUTHSTATUS | Performance Monitors Authentication Status register |
| External | PMCCFILTR_EL0 | Performance Monitors Cycle Counter Filter Register |
| External | PMCCNTR_EL0 | Performance Monitors Cycle Counter |
| External | PMCEID0 | Performance Monitors Common Event Identification register 0 |
| External | PMCEID1 | Performance Monitors Common Event Identification register 1 |
| External | PMCEID2 | Performance Monitors Common Event Identification register 2 |
| External | PMCEID3 | Performance Monitors Common Event Identification register 3 |
| External | PMCFGR | Performance Monitors Configuration Register |
| External | PMCID1SR | CONTEXTIDR_EL1 Sample Register |
| External | PMCID2SR | CONTEXTIDR_EL2 Sample Register |
| External | PMCIDR0 | Performance Monitors Component Identification Register 0 |
| External | PMCIDR1 | Performance Monitors Component Identification Register 1 |
| External | PMCIDR2 | Performance Monitors Component Identification Register 2 |
| External | PMCIDR3 | Performance Monitors Component Identification Register 3 |
| External | PMCNTENCLR_EL0 | Performance Monitors Count Enable Clear register |
| External | PMCNTENSET_EL0 | Performance Monitors Count Enable Set register |
| External | PMCR_EL0 | Performance Monitors Control Register |
| External | PMDEVAFF0 | Performance Monitors Device Affinity register 0 |
| External | PMDEVAFF1 | Performance Monitors Device Affinity register 1 |
| External | PMDEVARCH | Performance Monitors Device Architecture register |
| External | PMDEVID | Performance Monitors Device ID register |
| External | PMDEVTYPE | Performance Monitors Device Type register |
| External | PMEVCNTR<n>_EL0 | Performance Monitors Event Count Registers |
| External | PMEVFILTR<n> | Performance Monitors Event Type Select Register <n> |
| External | PMEVTYPER<n>_EL0 | Performance Monitors Event Type Registers |
| External | PMINTENCLR_EL1 | Performance Monitors Interrupt Enable Clear register |
| External | PMINTENSET_EL1 | Performance Monitors Interrupt Enable Set register |
| External | PMITCTRL | Performance Monitors Integration mode Control register |
| External | PMLAR | Performance Monitors Lock Access Register |
| External | PMLSR | Performance Monitors Lock Status Register |
| External | PMMIR | Performance Monitors Machine Identification Register |
| External | PMOVSCLR_EL0 | Performance Monitors Overflow Flag Status Clear register |
| External | PMOVSSSET_EL0 | Performance Monitors Overflow Flag Status Set register |
| External | PMPCSR | Program Counter Sample Register |
| External | PMPIDR0 | Performance Monitors Peripheral Identification Register 0 |
| External | PMPIDR1 | Performance Monitors Peripheral Identification Register 1 |
| External | PMPIDR2 | Performance Monitors Peripheral Identification Register 2 |
| External | PMPIDR3 | Performance Monitors Peripheral Identification Register 3 |
| External | PMPIDR4 | Performance Monitors Peripheral Identification Register 4 |
| External | PMSWINC_EL0 | Performance Monitors Software Increment register |
| External | PMVIDSR | VMID Sample Register |

In the Reset functional group:

| Exec state | Name | Description |
|------------|---------------------------|---|
| AArch32 | HRMR | Hyp Reset Management Register |
| AArch32 | RMR | Reset Management Register |
| AArch32 | RVBAR | Reset Vector Base Address Register |
| AArch64 | RMR_EL1 | Reset Management Register (EL1) |
| AArch64 | RMR_EL2 | Reset Management Register (EL2) |
| AArch64 | RMR_EL3 | Reset Management Register (EL3) |
| AArch64 | RVBAR_EL1 | Reset Vector Base Address Register (if EL2 and EL3 not implemented) |
| AArch64 | RVBAR_EL2 | Reset Vector Base Address Register (if EL3 not implemented) |
| AArch64 | RVBAR_EL3 | Reset Vector Base Address Register (if EL3 implemented) |

In the Thread functional group:

| Exec state | Name | Description |
|------------|-----------------------------|--|
| AArch32 | HTPIDR | Hyp Software Thread ID Register |
| AArch32 | TPIDRPRW | PL1 Software Thread ID Register |
| AArch32 | TPIDRURO | PL0 Read-Only Software Thread ID Register |
| AArch32 | TPIDRURW | PL0 Read/Write Software Thread ID Register |
| AArch64 | SCXTNUM_EL0 | EL0 Read/Write Software Context Number |
| AArch64 | SCXTNUM_EL1 | EL1 Read/Write Software Context Number |
| AArch64 | SCXTNUM_EL2 | EL2 Read/Write Software Context Number |
| AArch64 | SCXTNUM_EL3 | EL3 Read/Write Software Context Number |
| AArch64 | TPIDR2_EL0 | EL0 Read/Write Software Thread ID Register 2 |
| AArch64 | TPIDRRO_EL0 | EL0 Read-Only Software Thread ID Register |
| AArch64 | TPIDR_EL0 | EL0 Read/Write Software Thread ID Register |
| AArch64 | TPIDR_EL1 | EL1 Software Thread ID Register |
| AArch64 | TPIDR_EL2 | EL2 Software Thread ID Register |
| AArch64 | TPIDR_EL3 | EL3 Software Thread ID Register |

In the IMP DEF functional group:

| Exec state | Name | Description |
|------------|---|---|
| AArch32 | ACTLR | Auxiliary Control Register |
| AArch32 | ACTLR2 | Auxiliary Control Register 2 |
| AArch32 | ADFSR | Auxiliary Data Fault Status Register |
| AArch32 | AIDR | Auxiliary ID Register |
| AArch32 | AIFSR | Auxiliary Instruction Fault Status Register |
| AArch32 | AMAIR0 | Auxiliary Memory Attribute Indirection Register 0 |
| AArch32 | AMAIR1 | Auxiliary Memory Attribute Indirection Register 1 |
| AArch32 | HACTLR | Hyp Auxiliary Control Register |
| AArch32 | HACTLR2 | Hyp Auxiliary Control Register 2 |
| AArch32 | HADEFSR | Hyp Auxiliary Data Fault Status Register |
| AArch32 | HAIFSR | Hyp Auxiliary Instruction Fault Status Register |
| AArch32 | HAMAIR0 | Hyp Auxiliary Memory Attribute Indirection Register 0 |
| AArch32 | HAMAIR1 | Hyp Auxiliary Memory Attribute Indirection Register 1 |
| AArch64 | ACTLR_EL1 | Auxiliary Control Register (EL1) |
| AArch64 | ACTLR_EL2 | Auxiliary Control Register (EL2) |
| AArch64 | ACTLR_EL3 | Auxiliary Control Register (EL3) |
| AArch64 | AFSR0_EL1 | Auxiliary Fault Status Register 0 (EL1) |
| AArch64 | AFSR0_EL2 | Auxiliary Fault Status Register 0 (EL2) |
| AArch64 | AFSR0_EL3 | Auxiliary Fault Status Register 0 (EL3) |
| AArch64 | AFSR1_EL1 | Auxiliary Fault Status Register 1 (EL1) |
| AArch64 | AFSR1_EL2 | Auxiliary Fault Status Register 1 (EL2) |
| AArch64 | AFSR1_EL3 | Auxiliary Fault Status Register 1 (EL3) |
| AArch64 | AIDR_EL1 | Auxiliary ID Register |
| AArch64 | AMAIR_EL1 | Auxiliary Memory Attribute Indirection Register (EL1) |
| AArch64 | AMAIR_EL2 | Auxiliary Memory Attribute Indirection Register (EL2) |
| AArch64 | AMAIR_EL3 | Auxiliary Memory Attribute Indirection Register (EL3) |
| AArch64 | HACR_EL2 | Hypervisor Auxiliary Control Register |
| AArch64 | S3 <op1> <Cn> <Cm> <op2> | IMPLEMENTATION DEFINED registers |
| AArch64 | SYS S1 <op1> <Cn> <Cm> <op2>, SYSL S1 <op1> <Cn> <Cm> <op2> | IMPLEMENTATION DEFINED maintenance instructions |

In the Timer functional group:

| Exec state | Name | Description |
|------------|---------------------------------|---|
| AArch32 | CNTFRQ | Counter-timer Frequency register |
| AArch32 | CNTHPS_CTL | Counter-timer Secure Physical Timer Control Register (EL2) |
| AArch32 | CNTHPS_CVAL | Counter-timer Secure Physical Timer CompareValue Register (EL2) |
| AArch32 | CNTHPS_TVAL | Counter-timer Secure Physical Timer TimerValue Register (EL2) |
| AArch32 | CNTHP_CTL | Counter-timer Hyp Physical Timer Control register |
| AArch32 | CNTHVS_CTL | Counter-timer Secure Virtual Timer Control Register (EL2) |
| AArch32 | CNTHVS_CVAL | Counter-timer Secure Virtual Timer CompareValue Register (EL2) |
| AArch32 | CNTHVS_TVAL | Counter-timer Secure Virtual Timer TimerValue Register (EL2) |
| AArch32 | CNTHV_CTL | Counter-timer Virtual Timer Control register (EL2) |
| AArch32 | CNTHV_CVAL | Counter-timer Virtual Timer CompareValue register (EL2) |
| AArch32 | CNTHV_TVAL | Counter-timer Virtual Timer TimerValue register (EL2) |
| AArch32 | CNTKCTL | Counter-timer Kernel Control register |
| AArch32 | CNTPCT | Counter-timer Physical Count register |
| AArch32 | CNTPCTSS | Counter-timer Self-Synchronized Physical Count register |
| AArch32 | CNTP_CTL | Counter-timer Physical Timer Control register |
| AArch32 | CNTP_CVAL | Counter-timer Physical Timer CompareValue register |
| AArch32 | CNTP_TVAL | Counter-timer Physical Timer TimerValue register |
| AArch32 | CNTVCT | Counter-timer Virtual Count register |
| AArch32 | CNTVCTSS | Counter-timer Self-Synchronized Virtual Count register |
| AArch32 | CNTV_CTL | Counter-timer Virtual Timer Control register |
| AArch32 | CNTV_CVAL | Counter-timer Virtual Timer CompareValue register |
| AArch32 | CNTV_TVAL | Counter-timer Virtual Timer TimerValue register |
| AArch64 | CNTFRQ_EL0 | Counter-timer Frequency register |
| AArch64 | CNTHVS_CTL_EL2 | Counter-timer Secure Virtual Timer Control register (EL2) |
| AArch64 | CNTHVS_CVAL_EL2 | Counter-timer Secure Virtual Timer CompareValue register (EL2) |
| AArch64 | CNTHVS_TVAL_EL2 | Counter-timer Secure Virtual Timer TimerValue register (EL2) |
| AArch64 | CNTHV_CTL_EL2 | Counter-timer Virtual Timer Control register (EL2) |
| AArch64 | CNTHV_CVAL_EL2 | Counter-timer Virtual Timer CompareValue register (EL2) |
| AArch64 | CNTHV_TVAL_EL2 | Counter-timer Virtual Timer TimerValue Register (EL2) |
| AArch64 | CNTKCTL_EL1 | Counter-timer Kernel Control register |
| AArch64 | CNTPCTSS_EL0 | Counter-timer Self-Synchronized Physical Count register |
| AArch64 | CNTPCT_EL0 | Counter-timer Physical Count register |
| AArch64 | CNTPOFF_EL2 | Counter-timer Physical Offset register |
| AArch64 | CNTPS_CTL_EL1 | Counter-timer Physical Secure Timer Control register |
| AArch64 | CNTPS_CVAL_EL1 | Counter-timer Physical Secure Timer CompareValue register |
| AArch64 | CNTPS_TVAL_EL1 | Counter-timer Physical Secure Timer TimerValue register |
| AArch64 | CNTP_CTL_EL0 | Counter-timer Physical Timer Control register |
| AArch64 | CNTP_CVAL_EL0 | Counter-timer Physical Timer CompareValue register |
| AArch64 | CNTP_TVAL_EL0 | Counter-timer Physical Timer TimerValue register |
| AArch64 | CNTVCTSS_EL0 | Counter-timer Self-Synchronized Virtual Count register |
| AArch64 | CNTVCT_EL0 | Counter-timer Virtual Count register |
| AArch64 | CNTV_CTL_EL0 | Counter-timer Virtual Timer Control register |
| AArch64 | CNTV_CVAL_EL0 | Counter-timer Virtual Timer CompareValue register |
| AArch64 | CNTV_TVAL_EL0 | Counter-timer Virtual Timer TimerValue register |
| External | CNTACR<n> | Counter-timer Access Control Registers |
| External | CNTCR | Counter Control Register |
| External | CNTCV | Counter Count Value register |
| External | CNTELOACR | Counter-timer EL0 Access Control Register |
| External | CNTFID0 | Counter Frequency ID |
| External | CNTFID<n> | Counter Frequency IDs, n > 0 |
| External | CNTFRQ | Counter-timer Frequency |
| External | CNTID | Counter Identification Register |
| External | CNTNSAR | Counter-timer Non-secure Access Register |
| External | CNTPCT | Counter-timer Physical Count |
| External | CNTP_CTL | Counter-timer Physical Timer Control |
| External | CNTP_CVAL | Counter-timer Physical Timer CompareValue |
| External | CNTP_TVAL | Counter-timer Physical Timer TimerValue |
| External | CNTSCR | Counter Scale Register |
| External | CNTSR | Counter Status Register |
| External | CNTTIDR | Counter-timer Timer ID Register |
| External | CNTVCT | Counter-timer Virtual Count |
| External | CNTVOFF | Counter-timer Virtual Offset |

| Exec state | Name | Description |
|------------|------------------------------------|--|
| External | CNTVOFF<n> | Counter-timer Virtual Offsets |
| External | CNTV_CTL | Counter-timer Virtual Timer Control |
| External | CNTV_CVAL | Counter-timer Virtual Timer CompareValue |
| External | CNTV_TVAL | Counter-timer Virtual Timer TimerValue |
| External | CounterID<n> | Counter ID registers |

In the Debug functional group:

| Exec state | Name | Description |
|------------|-------------------------------------|--|
| AArch32 | DBGAUTHSTATUS | Debug Authentication Status register |
| AArch32 | DBGBCR<n> | Debug Breakpoint Control Registers |
| AArch32 | DBGBVR<n> | Debug Breakpoint Value Registers |
| AArch32 | DBGBXVR<n> | Debug Breakpoint Extended Value Registers |
| AArch32 | DBGCLAIMCLR | Debug CLAIM Tag Clear register |
| AArch32 | DBGCLAIMSET | Debug CLAIM Tag Set register |
| AArch32 | DBGDCCINT | DCC Interrupt Enable Register |
| AArch32 | DBGDEVID | Debug Device ID register 0 |
| AArch32 | DBGDEVID1 | Debug Device ID register 1 |
| AArch32 | DBGDEVID2 | Debug Device ID register 2 |
| AArch32 | DBGDIDR | Debug ID Register |
| AArch32 | DBGDRAR | Debug ROM Address Register |
| AArch32 | DBGDSAR | Debug Self Address Register |
| AArch32 | DBGDSCRext | Debug Status and Control Register, External View |
| AArch32 | DBGDSCRint | Debug Status and Control Register, Internal View |
| AArch32 | DBGDTRRXext | Debug OS Lock Data Transfer Register, Receive, External View |
| AArch32 | DBGDTRRXint | Debug Data Transfer Register, Receive |
| AArch32 | DBGDTRTXext | Debug OS Lock Data Transfer Register, Transmit |
| AArch32 | DBGDTRTXint | Debug Data Transfer Register, Transmit |
| AArch32 | DBGOSDLR | Debug OS Double Lock Register |
| AArch32 | DBGOSECCR | Debug OS Lock Exception Catch Control Register |
| AArch32 | DBGOSLAR | Debug OS Lock Access Register |
| AArch32 | DBGOSLSR | Debug OS Lock Status Register |
| AArch32 | DBGPRCR | Debug Power Control Register |
| AArch32 | DBGVCR | Debug Vector Catch Register |
| AArch32 | DBGWCR<n> | Debug Watchpoint Control Registers |
| AArch32 | DBGWFAR | Debug Watchpoint Fault Address Register |
| AArch32 | DBGWVR<n> | Debug Watchpoint Value Registers |
| AArch32 | TRFCR | Trace Filter Control Register |
| AArch64 | DBGAUTHSTATUS_EL1 | Debug Authentication Status register |
| AArch64 | DBGBCR<n>_EL1 | Debug Breakpoint Control Registers |
| AArch64 | DBGBVR<n>_EL1 | Debug Breakpoint Value Registers |
| AArch64 | DBGCLAIMCLR_EL1 | Debug CLAIM Tag Clear register |
| AArch64 | DBGCLAIMSET_EL1 | Debug CLAIM Tag Set register |
| AArch64 | DBGDTRRX_EL0 | Debug Data Transfer Register, Receive |
| AArch64 | DBGDTRTX_EL0 | Debug Data Transfer Register, Transmit |
| AArch64 | DBGDTR_EL0 | Debug Data Transfer Register, half-duplex |
| AArch64 | DBGPRCR_EL1 | Debug Power Control Register |
| AArch64 | DBGVCR32_EL2 | Debug Vector Catch Register |
| AArch64 | DBGWCR<n>_EL1 | Debug Watchpoint Control Registers |
| AArch64 | DBGWVR<n>_EL1 | Debug Watchpoint Value Registers |
| AArch64 | DLR_EL0 | Debug Link Register |
| AArch64 | DSPSR_EL0 | Debug Saved Program Status Register |
| AArch64 | MDCCINT_EL1 | Monitor DCC Interrupt Enable Register |
| AArch64 | MDCCSR_EL0 | Monitor DCC Status Register |
| AArch64 | MDRAR_EL1 | Monitor Debug ROM Address Register |
| AArch64 | MDSCR_EL1 | Monitor Debug System Control Register |
| AArch64 | OSDLR_EL1 | OS Double Lock Register |
| AArch64 | OSDTRRX_EL1 | OS Lock Data Transfer Register, Receive |
| AArch64 | OSDTRTX_EL1 | OS Lock Data Transfer Register, Transmit |
| AArch64 | OSECCR_EL1 | OS Lock Exception Catch Control Register |
| AArch64 | OSLAR_EL1 | OS Lock Access Register |
| AArch64 | OSLSR_EL1 | OS Lock Status Register |
| AArch64 | TRFCR_EL1 | Trace Filter Control Register (EL1) |

| Exec state | Name | Description |
|------------|-------------------------------------|---|
| AArch64 | TRFCR_EL2 | Trace Filter Control Register (EL2) |
| External | DBGAUTHSTATUS_EL1 | Debug Authentication Status register |
| External | DBGBCR<n>_EL1 | Debug Breakpoint Control Registers |
| External | DBGBVR<n>_EL1 | Debug Breakpoint Value Registers |
| External | DBGCLAIMCLR_EL1 | Debug CLAIM Tag Clear register |
| External | DBGCLAIMSET_EL1 | Debug CLAIM Tag Set register |
| External | DBGDTRRX_EL0 | Debug Data Transfer Register, Receive |
| External | DBGDTRTX_EL0 | Debug Data Transfer Register, Transmit |
| External | DBGWCR<n>_EL1 | Debug Watchpoint Control Registers |
| External | DBGWVR<n>_EL1 | Debug Watchpoint Value Registers |
| External | EDACR | External Debug Auxiliary Control Register |
| External | EDCIDR0 | External Debug Component Identification Register 0 |
| External | EDCIDR1 | External Debug Component Identification Register 1 |
| External | EDCIDR2 | External Debug Component Identification Register 2 |
| External | EDCIDR3 | External Debug Component Identification Register 3 |
| External | EDCIDSR | External Debug Context ID Sample Register |
| External | EDDEVAFF0 | External Debug Device Affinity register 0 |
| External | EDDEVAFF1 | External Debug Device Affinity register 1 |
| External | EDDEVARCH | External Debug Device Architecture register |
| External | EDDEVID | External Debug Device ID register 0 |
| External | EDDEVID1 | External Debug Device ID register 1 |
| External | EDDEVID2 | External Debug Device ID register 2 |
| External | EDDEVTYPE | External Debug Device Type register |
| External | EDECCR | External Debug Exception Catch Control Register |
| External | EDECR | External Debug Execution Control Register |
| External | EDES | External Debug Event Status Register |
| External | EDITCTRL | External Debug Integration mode Control register |
| External | EDITR | External Debug Instruction Transfer Register |
| External | EDLAR | External Debug Lock Access Register |
| External | EDLSR | External Debug Lock Status Register |
| External | EDPCSR | External Debug Program Counter Sample Register |
| External | EDPIDR0 | External Debug Peripheral Identification Register 0 |
| External | EDPIDR1 | External Debug Peripheral Identification Register 1 |
| External | EDPIDR2 | External Debug Peripheral Identification Register 2 |
| External | EDPIDR3 | External Debug Peripheral Identification Register 3 |
| External | EDPIDR4 | External Debug Peripheral Identification Register 4 |
| External | EDPRCR | External Debug Power/Reset Control Register |
| External | EDPRSR | External Debug Processor Status Register |
| External | EDRCR | External Debug Reserve Control Register |
| External | EDSCR | External Debug Status and Control Register |
| External | EDVIDSR | External Debug Virtual Context Sample Register |
| External | EDWAR | External Debug Watchpoint Address Register |
| External | OSLAR_EL1 | OS Lock Access Register |

In the CTI functional group:

| Exec state | Name | Description |
|------------|--------------------------------|---|
| External | ASICCTL | CTI External Multiplexer Control register |
| External | CTIAPPCLEAR | CTI Application Trigger Clear register |
| External | CTIAPPPULSE | CTI Application Pulse register |
| External | CTIAPPSET | CTI Application Trigger Set register |
| External | CTIAUTHSTATUS | CTI Authentication Status register |
| External | CTICHINSTATUS | CTI Channel In Status register |
| External | CTICHOUTSTATUS | CTI Channel Out Status register |
| External | CTICIDR0 | CTI Component Identification Register 0 |
| External | CTICIDR1 | CTI Component Identification Register 1 |
| External | CTICIDR2 | CTI Component Identification Register 2 |
| External | CTICIDR3 | CTI Component Identification Register 3 |
| External | CTICLAIMCLR | CTI CLAIM Tag Clear register |
| External | CTICLAIMSET | CTI CLAIM Tag Set register |
| External | CTICONTROL | CTI Control register |
| External | CTIDEVAFF0 | CTI Device Affinity register 0 |
| External | CTIDEVAFF1 | CTI Device Affinity register 1 |

| Exec state | Name | Description |
|------------|-----------------------------------|--|
| External | CTIDEVARCH | CTI Device Architecture register |
| External | CTIDEVCTL | CTI Device Control register |
| External | CTIDEVID | CTI Device ID register 0 |
| External | CTIDEVID1 | CTI Device ID register 1 |
| External | CTIDEVID2 | CTI Device ID register 2 |
| External | CTIDEVTYPE | CTI Device Type register |
| External | CTIGATE | CTI Channel Gate Enable register |
| External | CTIINEN<n> | CTI Input Trigger to Output Channel Enable registers |
| External | CTIINTACK | CTI Output Trigger Acknowledge register |
| External | CTIITCTRL | CTI Integration mode Control register |
| External | CTILAR | CTI Lock Access Register |
| External | CTILSR | CTI Lock Status Register |
| External | CTIOUTEN<n> | CTI Input Channel to Output Trigger Enable registers |
| External | CTIPIDR0 | CTI Peripheral Identification Register 0 |
| External | CTIPIDR1 | CTI Peripheral Identification Register 1 |
| External | CTIPIDR2 | CTI Peripheral Identification Register 2 |
| External | CTIPIDR3 | CTI Peripheral Identification Register 3 |
| External | CTIPIDR4 | CTI Peripheral Identification Register 4 |
| External | CTITRIGINSTATUS | CTI Trigger In Status register |
| External | CTITRIGOUTSTATUS | CTI Trigger Out Status register |

In the Virt functional group:

| Exec state | Name | Description |
|------------|-----------------------------------|--|
| AArch32 | ATS1HR | Address Translate Stage 1 Hyp mode Read |
| AArch32 | ATS1HW | Address Translate Stage 1 Hyp mode Write |
| AArch32 | CNTHCTL | Counter-timer Hyp Control register |
| AArch32 | CNTHP_CVAL | Counter-timer Hyp Physical CompareValue register |
| AArch32 | CNTHP_TVAL | Counter-timer Hyp Physical Timer TimerValue register |
| AArch32 | CNTVOFF | Counter-timer Virtual Offset register |
| AArch32 | HACR | Hyp Auxiliary Configuration Register |
| AArch32 | HACTLR | Hyp Auxiliary Control Register |
| AArch32 | HACTLR2 | Hyp Auxiliary Control Register 2 |
| AArch32 | HADFSR | Hyp Auxiliary Data Fault Status Register |
| AArch32 | HAIFSR | Hyp Auxiliary Instruction Fault Status Register |
| AArch32 | HAMAIR0 | Hyp Auxiliary Memory Attribute Indirection Register 0 |
| AArch32 | HAMAIR1 | Hyp Auxiliary Memory Attribute Indirection Register 1 |
| AArch32 | HCPTR | Hyp Architectural Feature Trap Register |
| AArch32 | HCR | Hyp Configuration Register |
| AArch32 | HCR2 | Hyp Configuration Register 2 |
| AArch32 | HDCR | Hyp Debug Control Register |
| AArch32 | HDFAR | Hyp Data Fault Address Register |
| AArch32 | HIFAR | Hyp Instruction Fault Address Register |
| AArch32 | HMAIR0 | Hyp Memory Attribute Indirection Register 0 |
| AArch32 | HMAIR1 | Hyp Memory Attribute Indirection Register 1 |
| AArch32 | HPEAR | Hyp IPA Fault Address Register |
| AArch32 | HRMR | Hyp Reset Management Register |
| AArch32 | HSCTLR | Hyp System Control Register |
| AArch32 | HSR | Hyp Syndrome Register |
| AArch32 | HSTR | Hyp System Trap Register |
| AArch32 | HTCR | Hyp Translation Control Register |
| AArch32 | HTPIDR | Hyp Software Thread ID Register |
| AArch32 | HTREFCR | Hyp Trace Filter Control Register |
| AArch32 | HTTBR | Hyp Translation Table Base Register |
| AArch32 | HVBAR | Hyp Vector Base Address Register |
| AArch32 | ICC_HSRE | Interrupt Controller Hyp System Register Enable register |
| AArch32 | ICH_AP0R<n> | Interrupt Controller Hyp Active Priorities Group 0 Registers |
| AArch32 | ICH_AP1R<n> | Interrupt Controller Hyp Active Priorities Group 1 Registers |
| AArch32 | ICH_EISR | Interrupt Controller End of Interrupt Status Register |
| AArch32 | ICH_ELRSR | Interrupt Controller Empty List Register Status Register |
| AArch32 | ICH_HCR | Interrupt Controller Hyp Control Register |
| AArch32 | ICH_LR<n> | Interrupt Controller List Registers |

| Exec state | Name | Description |
|------------|---|---|
| AArch32 | ICH_LRC<n> | Interrupt Controller List Registers |
| AArch32 | ICH_MISR | Interrupt Controller Maintenance Interrupt State Register |
| AArch32 | ICH_VMCR | Interrupt Controller Virtual Machine Control Register |
| AArch32 | ICH_VTR | Interrupt Controller VGIC Type Register |
| AArch32 | TLBIALLH | TLB Invalidate All, Hyp mode |
| AArch32 | TLBIALLHIS | TLB Invalidate All, Hyp mode, Inner Shareable |
| AArch32 | TLBIIPAS2 | TLB Invalidate by Intermediate Physical Address, Stage 2 |
| AArch32 | TLBIIPAS2IS | TLB Invalidate by Intermediate Physical Address, Stage 2, Inner Shareable |
| AArch32 | TLBIIPAS2L | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level |
| AArch32 | TLBIIPAS2LIS | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, Inner Shareable |
| AArch32 | TLBIMVAH | TLB Invalidate by VA, Hyp mode |
| AArch32 | TLBIMVAHIS | TLB Invalidate by VA, Hyp mode, Inner Shareable |
| AArch32 | TLBIMVALH | TLB Invalidate by VA, Last level, Hyp mode |
| AArch32 | TLBIMVALHIS | TLB Invalidate by VA, Last level, Hyp mode, Inner Shareable |
| AArch32 | VMPIDR | Virtualization Multiprocessor ID Register |
| AArch32 | VPIDR | Virtualization Processor ID Register |
| AArch32 | VTCR | Virtualization Translation Control Register |
| AArch32 | VTTBR | Virtualization Translation Table Base Register |
| AArch64 | ACTLR_EL2 | Auxiliary Control Register (EL2) |
| AArch64 | AFSR0_EL2 | Auxiliary Fault Status Register 0 (EL2) |
| AArch64 | AFSR1_EL2 | Auxiliary Fault Status Register 1 (EL2) |
| AArch64 | AMAIR_EL2 | Auxiliary Memory Attribute Indirection Register (EL2) |
| AArch64 | CNTHCTL_EL2 | Counter-timer Hypervisor Control register |
| AArch64 | CNTHPS_CTL_EL2 | Counter-timer Secure Physical Timer Control register (EL2) |
| AArch64 | CNTHPS_CVAL_EL2 | Counter-timer Secure Physical Timer CompareValue register (EL2) |
| AArch64 | CNTHPS_TVAL_EL2 | Counter-timer Secure Physical Timer TimerValue register (EL2) |
| AArch64 | CNTHP_CTL_EL2 | Counter-timer Hypervisor Physical Timer Control register |
| AArch64 | CNTHP_CVAL_EL2 | Counter-timer Physical Timer CompareValue register (EL2) |
| AArch64 | CNTHP_TVAL_EL2 | Counter-timer Physical Timer TimerValue register (EL2) |
| AArch64 | CNTVOFF_EL2 | Counter-timer Virtual Offset register |
| AArch64 | CPTR_EL2 | Architectural Feature Trap Register (EL2) |
| AArch64 | ESR_EL2 | Exception Syndrome Register (EL2) |
| AArch64 | FAR_EL2 | Fault Address Register (EL2) |
| AArch64 | HACR_EL2 | Hypervisor Auxiliary Control Register |
| AArch64 | HCRX_EL2 | Extended Hypervisor Configuration Register |
| AArch64 | HCR_EL2 | Hypervisor Configuration Register |
| AArch64 | HPFAR_EL2 | Hypervisor IPA Fault Address Register |
| AArch64 | HSTR_EL2 | Hypervisor System Trap Register |
| AArch64 | ICC_SRE_EL2 | Interrupt Controller System Register Enable register (EL2) |
| AArch64 | ICH_AP0R<n>_EL2 | Interrupt Controller Hyp Active Priorities Group 0 Registers |
| AArch64 | ICH_AP1R<n>_EL2 | Interrupt Controller Hyp Active Priorities Group 1 Registers |
| AArch64 | ICH_EISR_EL2 | Interrupt Controller End of Interrupt Status Register |
| AArch64 | ICH_ELSR_EL2 | Interrupt Controller Empty List Register Status Register |
| AArch64 | ICH_HCR_EL2 | Interrupt Controller Hyp Control Register |
| AArch64 | ICH_LR<n>_EL2 | Interrupt Controller List Registers |
| AArch64 | ICH_MISR_EL2 | Interrupt Controller Maintenance Interrupt State Register |
| AArch64 | ICH_VMCR_EL2 | Interrupt Controller Virtual Machine Control Register |
| AArch64 | ICH_VTR_EL2 | Interrupt Controller VGIC Type Register |
| AArch64 | MAIR_EL2 | Memory Attribute Indirection Register (EL2) |
| AArch64 | MDCR_EL2 | Monitor Debug Configuration Register (EL2) |
| AArch64 | RMR_EL2 | Reset Management Register (EL2) |
| AArch64 | SCTLR_EL2 | System Control Register (EL2) |
| AArch64 | TCR_EL2 | Translation Control Register (EL2) |
| AArch64 | TLBI_IPAS2E1, TLBI_IPAS2E1NXS | TLB Invalidate by Intermediate Physical Address, Stage 2, EL1 |
| AArch64 | TLBI_IPAS2E1IS, TLBI_IPAS2E1ISNXS | TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable |
| AArch64 | TLBI_IPAS2E1OS, TLBI_IPAS2E1OSNXS | TLB Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable |
| AArch64 | TLBI_IPAS2LE1, TLBI_IPAS2LE1NXS | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1 |

| Exec state | Name | Description |
|------------|---|--|
| AArch64 | TLBI IPAS2LE1IS, TLBI IPAS2LE1ISNXS | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable |
| AArch64 | TLBI IPAS2LE1IOS, TLBI IPAS2LE1IOSNXS | TLB Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable |
| AArch64 | TLBI RIPAS2E1, TLBI RIPAS2E1NXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1 |
| AArch64 | TLBI RIPAS2E1IS, TLBI RIPAS2E1ISNXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Inner Shareable |
| AArch64 | TLBI RIPAS2E1IOS, TLBI RIPAS2E1IOSNXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, EL1, Outer Shareable |
| AArch64 | TLBI RIPAS2LE1, TLBI RIPAS2LE1NXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1 |
| AArch64 | TLBI RIPAS2LE1IS, TLBI RIPAS2LE1ISNXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Inner Shareable |
| AArch64 | TLBI RIPAS2LE1IOS, TLBI RIPAS2LE1IOSNXS | TLB Range Invalidate by Intermediate Physical Address, Stage 2, Last level, EL1, Outer Shareable |
| AArch64 | TPIDR_EL2 | EL2 Software Thread ID Register |
| AArch64 | TTBR0_EL2 | Translation Table Base Register 0 (EL2) |
| AArch64 | TTBR1_EL2 | Translation Table Base Register 1 (EL2) |
| AArch64 | VBAR_EL2 | Vector Base Address Register (EL2) |
| AArch64 | VMPIDR_EL2 | Virtualization Multiprocessor ID Register |
| AArch64 | VPIDR_EL2 | Virtualization Processor ID Register |
| AArch64 | VTCR_EL2 | Virtualization Translation Control Register |
| AArch64 | VTTBR_EL2 | Virtualization Translation Table Base Register |

In the Secure functional group:

| Exec state | Name | Description |
|------------|------------------------------|--|
| AArch32 | ICC_MCTLR | Interrupt Controller Monitor Control Register |
| AArch32 | ICC_MSRE | Interrupt Controller Monitor System Register Enable register |
| AArch32 | MVBAR | Monitor Vector Base Address Register |
| AArch32 | NSACR | Non-Secure Access Control Register |
| AArch32 | SCR | Secure Configuration Register |
| AArch32 | SDCR | Secure Debug Control Register |
| AArch32 | SDER | Secure Debug Enable Register |
| AArch64 | ACTLR_EL3 | Auxiliary Control Register (EL3) |
| AArch64 | AFSR0_EL3 | Auxiliary Fault Status Register 0 (EL3) |
| AArch64 | AFSR1_EL3 | Auxiliary Fault Status Register 1 (EL3) |
| AArch64 | AMAIR_EL3 | Auxiliary Memory Attribute Indirection Register (EL3) |
| AArch64 | CPTR_EL3 | Architectural Feature Trap Register (EL3) |
| AArch64 | ICC_CTLR_EL3 | Interrupt Controller Control Register (EL3) |
| AArch64 | ICC_SRE_EL3 | Interrupt Controller System Register Enable register (EL3) |
| AArch64 | MDCR_EL3 | Monitor Debug Configuration Register (EL3) |
| AArch64 | SCR_EL3 | Secure Configuration Register |
| AArch64 | SDER32_EL3 | AArch32 Secure Debug Enable Register |
| AArch64 | VBAR_EL3 | Vector Base Address Register (EL3) |

In the Float functional group:

| Exec state | Name | Description |
|------------|-----------------------------|--|
| AArch32 | FPExc | Floating-Point Exception Control register |
| AArch32 | FPSCR | Floating-Point Status and Control Register |
| AArch32 | FPSID | Floating-Point System ID register |
| AArch32 | MVFR0 | Media and VFP Feature Register 0 |
| AArch32 | MVFR1 | Media and VFP Feature Register 1 |
| AArch32 | MVFR2 | Media and VFP Feature Register 2 |
| AArch64 | FPCR | Floating-point Control Register |
| AArch64 | FPExc32_EL2 | Floating-Point Exception Control register |
| AArch64 | FPSR | Floating-point Status Register |
| AArch64 | MVFR0_EL1 | AArch32 Media and VFP Feature Register 0 |
| AArch64 | MVFR1_EL1 | AArch32 Media and VFP Feature Register 1 |
| AArch64 | MVFR2_EL1 | AArch32 Media and VFP Feature Register 2 |

In the Legacy functional group:

| Exec state | Name | Description |
|------------|-------------------------|--|
| AArch32 | CP15DMB | Data Memory Barrier System instruction |
| AArch32 | CP15DSB | Data Synchronization Barrier System instruction |
| AArch32 | CP15ISB | Instruction Synchronization Barrier System instruction |
| AArch32 | FCSEIDR | FCSE Process ID register |
| AArch32 | JIDR | Jazelle ID Register |
| AArch32 | JMCR | Jazelle Main Configuration Register |
| AArch32 | JOSCR | Jazelle OS Control Register |

In the Trace functional group:

| Exec state | Name | Description |
|------------|---------------------------------------|--|
| AArch64 | TRCACATR<n> | Address Comparator Access Type Register <n> |
| AArch64 | TRCACVR<n> | Address Comparator Value Register <n> |
| AArch64 | TRCAUXCLR | Auxiliary Control Register |
| AArch64 | TRCBBCTLR | Branch Broadcast Control Register |
| AArch64 | TRCCCCLR | Cycle Count Control Register |
| AArch64 | TRCCIDCCTLR0 | Context Identifier Comparator Control Register 0 |
| AArch64 | TRCCIDCCTLR1 | Context Identifier Comparator Control Register 1 |
| AArch64 | TRCCIDCVR<n> | Context Identifier Comparator Value Registers <n> |
| AArch64 | TRCCLAIMCLR | Claim Tag Clear Register |
| AArch64 | TRCCLAIMSET | Claim Tag Set Register |
| AArch64 | TRCCNTCTLR<n> | Counter Control Register <n> |
| AArch64 | TRCCNTRLDVR<n> | Counter Reload Value Register <n> |
| AArch64 | TRCCNTVR<n> | Counter Value Register <n> |
| AArch64 | TRCCONFIGR | Trace Configuration Register |
| AArch64 | TRCEVENTCTL0R | Event Control 0 Register |
| AArch64 | TRCEVENTCTL1R | Event Control 1 Register |
| AArch64 | TRCEXTINSELR<n> | External Input Select Register <n> |
| AArch64 | TRCIDR0 | ID Register 0 |
| AArch64 | TRCIDR1 | ID Register 1 |
| AArch64 | TRCIDR10 | ID Register 10 |
| AArch64 | TRCIDR11 | ID Register 11 |
| AArch64 | TRCIDR12 | ID Register 12 |
| AArch64 | TRCIDR13 | ID Register 13 |
| AArch64 | TRCIDR2 | ID Register 2 |
| AArch64 | TRCIDR3 | ID Register 3 |
| AArch64 | TRCIDR4 | ID Register 4 |
| AArch64 | TRCIDR5 | ID Register 5 |
| AArch64 | TRCIDR6 | ID Register 6 |
| AArch64 | TRCIDR7 | ID Register 7 |
| AArch64 | TRCIDR8 | ID Register 8 |
| AArch64 | TRCIDR9 | ID Register 9 |
| AArch64 | TRCIMSPEC0 | IMP DEF Register 0 |
| AArch64 | TRCIMSPEC<n> | IMP DEF Register <n> |
| AArch64 | TRCPRGCTLR | Programming Control Register |
| AArch64 | TRCQCTLR | Q Element Control Register |
| AArch64 | TRCRSCTLR<n> | Resource Selection Control Register <n> |
| AArch64 | TRCRSR | Resources Status Register |
| AArch64 | TRCSEQEVR<n> | Sequencer State Transition Control Register <n> |
| AArch64 | TRCSEQRSTVR | Sequencer Reset Control Register |
| AArch64 | TRCSEQSTR | Sequencer State Register |
| AArch64 | TRCSSCCR<n> | Single-shot Comparator Control Register <n> |
| AArch64 | TRCSSCSR<n> | Single-shot Comparator Control Status Register <n> |
| AArch64 | TRCSSPCICR<n> | Single-shot Processing Element Comparator Input Control Register <n> |
| AArch64 | TRCSTALLCTLR | Stall Control Register |
| AArch64 | TRCSTATR | Trace Status Register |
| AArch64 | TRCSYNCPR | Synchronization Period Register |
| AArch64 | TRCTRAIDR | Trace ID Register |
| AArch64 | TRCTSCTLR | Timestamp Control Register |
| AArch64 | TRCVICTLR | ViewInst Main Control Register |
| AArch64 | TRCVIIECTLR | ViewInst Include/Exclude Control Register |

| Exec state | Name | Description |
|------------|---------------------------------------|--|
| AArch64 | TRCVIPCSSLTR | ViewInst Start/Stop PE Comparator Control Register |
| AArch64 | TRCVISSCTRL | ViewInst Start/Stop Control Register |
| AArch64 | TRCVMIDCCTLR0 | Virtual Context Identifier Comparator Control Register 0 |
| AArch64 | TRCVMIDCCTLR1 | Virtual Context Identifier Comparator Control Register 1 |
| AArch64 | TRCVMIDCVR<n> | Virtual Context Identifier Comparator Value Register <n> |
| External | TRCACATR<n> | Address Comparator Access Type Register <n> |
| External | TRCACVR<n> | Address Comparator Value Register <n> |
| External | TRCAUXCTRL | Auxiliary Control Register |
| External | TRCBBCTRL | Branch Broadcast Control Register |
| External | TRCCCCTRL | Cycle Count Control Register |
| External | TRCCIDCCTLR0 | Context Identifier Comparator Control Register 0 |
| External | TRCCIDCCTLR1 | Context Identifier Comparator Control Register 1 |
| External | TRCCIDCVR<n> | Context Identifier Comparator Value Registers <n> |
| External | TRCCLAIMCLR | Claim Tag Clear Register |
| External | TRCCLAIMSET | Claim Tag Set Register |
| External | TRCCNTCTRL<n> | Counter Control Register <n> |
| External | TRCCNTRLDVR<n> | Counter Reload Value Register <n> |
| External | TRCCNTVR<n> | Counter Value Register <n> |
| External | TRCCONFIGR | Trace Configuration Register |
| External | TRCEVENTCTL0R | Event Control 0 Register |
| External | TRCEVENTCTL1R | Event Control 1 Register |
| External | TRCEXTINSELR<n> | External Input Select Register <n> |
| External | TRCIDR0 | ID Register 0 |
| External | TRCIDR1 | ID Register 1 |
| External | TRCIDR10 | ID Register 10 |
| External | TRCIDR11 | ID Register 11 |
| External | TRCIDR12 | ID Register 12 |
| External | TRCIDR13 | ID Register 13 |
| External | TRCIDR2 | ID Register 2 |
| External | TRCIDR3 | ID Register 3 |
| External | TRCIDR4 | ID Register 4 |
| External | TRCIDR5 | ID Register 5 |
| External | TRCIDR6 | ID Register 6 |
| External | TRCIDR7 | ID Register 7 |
| External | TRCIDR8 | ID Register 8 |
| External | TRCIDR9 | ID Register 9 |
| External | TRCIMSPEC0 | IMP DEF Register 0 |
| External | TRCIMSPEC<n> | IMP DEF Register <n> |
| External | TRCPRGCTRL | Programming Control Register |
| External | TRCQCTRL | Q Element Control Register |
| External | TRCRSCTRL<n> | Resource Selection Control Register <n> |
| External | TRCRSR | Resources Status Register |
| External | TRCSEQEVR<n> | Sequencer State Transition Control Register <n> |
| External | TRCSEQRSTEV | Sequencer Reset Control Register |
| External | TRCSEQSTR | Sequencer State Register |
| External | TRCSSCCR<n> | Single-shot Comparator Control Register <n> |
| External | TRCSSCSR<n> | Single-shot Comparator Control Status Register <n> |
| External | TRCSSPICR<n> | Single-shot Processing Element Comparator Input Control Register <n> |
| External | TRCSTALLCTRL | Stall Control Register |
| External | TRCSTATR | Trace Status Register |
| External | TRCSYNCP | Synchronization Period Register |
| External | TRCTRACEIDR | Trace ID Register |
| External | TRCTSCTRL | Timestamp Control Register |
| External | TRCVICTLR | ViewInst Main Control Register |
| External | TRCVIIECTRL | ViewInst Include/Exclude Control Register |
| External | TRCVIPCSSLTR | ViewInst Start/Stop PE Comparator Control Register |
| External | TRCVISSCTRL | ViewInst Start/Stop Control Register |
| External | TRCVMIDCCTLR0 | Virtual Context Identifier Comparator Control Register 0 |
| External | TRCVMIDCCTLR1 | Virtual Context Identifier Comparator Control Register 1 |
| External | TRCVMIDCVR<n> | Virtual Context Identifier Comparator Value Register <n> |

In the GIC functional group:

| Exec state | Name | Description |
|------------|---------------------------------------|--|
| AArch32 | ICC_AP0R<n> | Interrupt Controller Active Priorities Group 0 Registers |
| AArch32 | ICC_AP1R<n> | Interrupt Controller Active Priorities Group 1 Registers |
| AArch32 | ICC_ASGI1R | Interrupt Controller Alias Software Generated Interrupt Group 1 Register |
| AArch32 | ICC_BPR0 | Interrupt Controller Binary Point Register 0 |
| AArch32 | ICC_BPR1 | Interrupt Controller Binary Point Register 1 |
| AArch32 | ICC_CTLR | Interrupt Controller Control Register |
| AArch32 | ICC_DIR | Interrupt Controller Deactivate Interrupt Register |
| AArch32 | ICC_EOIR0 | Interrupt Controller End Of Interrupt Register 0 |
| AArch32 | ICC_EOIR1 | Interrupt Controller End Of Interrupt Register 1 |
| AArch32 | ICC_HPIR0 | Interrupt Controller Highest Priority Pending Interrupt Register 0 |
| AArch32 | ICC_HPIR1 | Interrupt Controller Highest Priority Pending Interrupt Register 1 |
| AArch32 | ICC_HSRE | Interrupt Controller Hyp System Register Enable register |
| AArch32 | ICC_IAR0 | Interrupt Controller Interrupt Acknowledge Register 0 |
| AArch32 | ICC_IAR1 | Interrupt Controller Interrupt Acknowledge Register 1 |
| AArch32 | ICC_IGRPEN0 | Interrupt Controller Interrupt Group 0 Enable register |
| AArch32 | ICC_IGRPEN1 | Interrupt Controller Interrupt Group 1 Enable register |
| AArch32 | ICC_MCTLR | Interrupt Controller Monitor Control Register |
| AArch32 | ICC_MGRPEN1 | Interrupt Controller Monitor Interrupt Group 1 Enable register |
| AArch32 | ICC_MSRE | Interrupt Controller Monitor System Register Enable register |
| AArch32 | ICC_PMR | Interrupt Controller Interrupt Priority Mask Register |
| AArch32 | ICC_RPR | Interrupt Controller Running Priority Register |
| AArch32 | ICC_SGI0R | Interrupt Controller Software Generated Interrupt Group 0 Register |
| AArch32 | ICC_SGI1R | Interrupt Controller Software Generated Interrupt Group 1 Register |
| AArch32 | ICC_SRE | Interrupt Controller System Register Enable register |
| AArch32 | ICH_AP0R<n> | Interrupt Controller Hyp Active Priorities Group 0 Registers |
| AArch32 | ICH_AP1R<n> | Interrupt Controller Hyp Active Priorities Group 1 Registers |
| AArch32 | ICH_EISR | Interrupt Controller End of Interrupt Status Register |
| AArch32 | ICH_ELRSR | Interrupt Controller Empty List Register Status Register |
| AArch32 | ICH_HCR | Interrupt Controller Hyp Control Register |
| AArch32 | ICH_LR<n> | Interrupt Controller List Registers |
| AArch32 | ICH_LRC<n> | Interrupt Controller List Registers |
| AArch32 | ICH_MISR | Interrupt Controller Maintenance Interrupt State Register |
| AArch32 | ICH_VMCR | Interrupt Controller Virtual Machine Control Register |
| AArch32 | ICH_VTR | Interrupt Controller VGIC Type Register |
| AArch32 | ICV_AP0R<n> | Interrupt Controller Virtual Active Priorities Group 0 Registers |
| AArch32 | ICV_AP1R<n> | Interrupt Controller Virtual Active Priorities Group 1 Registers |
| AArch32 | ICV_BPR0 | Interrupt Controller Virtual Binary Point Register 0 |
| AArch32 | ICV_BPR1 | Interrupt Controller Virtual Binary Point Register 1 |
| AArch32 | ICV_CTLR | Interrupt Controller Virtual Control Register |
| AArch32 | ICV_DIR | Interrupt Controller Deactivate Virtual Interrupt Register |
| AArch32 | ICV_EOIR0 | Interrupt Controller Virtual End Of Interrupt Register 0 |
| AArch32 | ICV_EOIR1 | Interrupt Controller Virtual End Of Interrupt Register 1 |
| AArch32 | ICV_HPIR0 | Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0 |
| AArch32 | ICV_HPIR1 | Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1 |
| AArch32 | ICV_IAR0 | Interrupt Controller Virtual Interrupt Acknowledge Register 0 |
| AArch32 | ICV_IAR1 | Interrupt Controller Virtual Interrupt Acknowledge Register 1 |
| AArch32 | ICV_IGRPEN0 | Interrupt Controller Virtual Interrupt Group 0 Enable register |
| AArch32 | ICV_IGRPEN1 | Interrupt Controller Virtual Interrupt Group 1 Enable register |
| AArch32 | ICV_PMR | Interrupt Controller Virtual Interrupt Priority Mask Register |
| AArch32 | ICV_RPR | Interrupt Controller Virtual Running Priority Register |
| AArch64 | ICC_AP0R<n>_EL1 | Interrupt Controller Active Priorities Group 0 Registers |
| AArch64 | ICC_AP1R<n>_EL1 | Interrupt Controller Active Priorities Group 1 Registers |
| AArch64 | ICC_ASGI1R_EL1 | Interrupt Controller Alias Software Generated Interrupt Group 1 Register |
| AArch64 | ICC_BPR0_EL1 | Interrupt Controller Binary Point Register 0 |
| AArch64 | ICC_BPR1_EL1 | Interrupt Controller Binary Point Register 1 |
| AArch64 | ICC_CTLR_EL1 | Interrupt Controller Control Register (EL1) |
| AArch64 | ICC_CTLR_EL3 | Interrupt Controller Control Register (EL3) |
| AArch64 | ICC_DIR_EL1 | Interrupt Controller Deactivate Interrupt Register |
| AArch64 | ICC_EOIR0_EL1 | Interrupt Controller End Of Interrupt Register 0 |
| AArch64 | ICC_EOIR1_EL1 | Interrupt Controller End Of Interrupt Register 1 |
| AArch64 | ICC_HPIR0_EL1 | Interrupt Controller Highest Priority Pending Interrupt Register 0 |
| AArch64 | ICC_HPIR1_EL1 | Interrupt Controller Highest Priority Pending Interrupt Register 1 |

| Exec state | Name | Description |
|------------|---------------------------------------|--|
| AArch64 | ICC_IAR0_EL1 | Interrupt Controller Interrupt Acknowledge Register 0 |
| AArch64 | ICC_IAR1_EL1 | Interrupt Controller Interrupt Acknowledge Register 1 |
| AArch64 | ICC_IGRPEN0_EL1 | Interrupt Controller Interrupt Group 0 Enable register |
| AArch64 | ICC_IGRPEN1_EL1 | Interrupt Controller Interrupt Group 1 Enable register |
| AArch64 | ICC_IGRPEN1_EL3 | Interrupt Controller Interrupt Group 1 Enable register (EL3) |
| AArch64 | ICC_NMIAR1_EL1 | Interrupt Controller Non-maskable Interrupt Acknowledge Register 1 |
| AArch64 | ICC_PMR_EL1 | Interrupt Controller Interrupt Priority Mask Register |
| AArch64 | ICC_RPR_EL1 | Interrupt Controller Running Priority Register |
| AArch64 | ICC_SGI0R_EL1 | Interrupt Controller Software Generated Interrupt Group 0 Register |
| AArch64 | ICC_SGI1R_EL1 | Interrupt Controller Software Generated Interrupt Group 1 Register |
| AArch64 | ICC_SRE_EL1 | Interrupt Controller System Register Enable register (EL1) |
| AArch64 | ICC_SRE_EL2 | Interrupt Controller System Register Enable register (EL2) |
| AArch64 | ICC_SRE_EL3 | Interrupt Controller System Register Enable register (EL3) |
| AArch64 | ICH_AP0R<n>_EL2 | Interrupt Controller Hyp Active Priorities Group 0 Registers |
| AArch64 | ICH_AP1R<n>_EL2 | Interrupt Controller Hyp Active Priorities Group 1 Registers |
| AArch64 | ICH_EISR_EL2 | Interrupt Controller End of Interrupt Status Register |
| AArch64 | ICH_ELRSR_EL2 | Interrupt Controller Empty List Register Status Register |
| AArch64 | ICH_HCR_EL2 | Interrupt Controller Hyp Control Register |
| AArch64 | ICH_LR<n>_EL2 | Interrupt Controller List Registers |
| AArch64 | ICH_MISR_EL2 | Interrupt Controller Maintenance Interrupt State Register |
| AArch64 | ICH_VMCR_EL2 | Interrupt Controller Virtual Machine Control Register |
| AArch64 | ICH_VTR_EL2 | Interrupt Controller VGIC Type Register |
| AArch64 | ICV_AP0R<n>_EL1 | Interrupt Controller Virtual Active Priorities Group 0 Registers |
| AArch64 | ICV_AP1R<n>_EL1 | Interrupt Controller Virtual Active Priorities Group 1 Registers |
| AArch64 | ICV_BPR0_EL1 | Interrupt Controller Virtual Binary Point Register 0 |
| AArch64 | ICV_BPR1_EL1 | Interrupt Controller Virtual Binary Point Register 1 |
| AArch64 | ICV_CTLR_EL1 | Interrupt Controller Virtual Control Register |
| AArch64 | ICV_DIR_EL1 | Interrupt Controller Deactivate Virtual Interrupt Register |
| AArch64 | ICV_EOIR0_EL1 | Interrupt Controller Virtual End Of Interrupt Register 0 |
| AArch64 | ICV_EOIR1_EL1 | Interrupt Controller Virtual End Of Interrupt Register 1 |
| AArch64 | ICV_HPPIR0_EL1 | Interrupt Controller Virtual Highest Priority Pending Interrupt Register 0 |
| AArch64 | ICV_HPPIR1_EL1 | Interrupt Controller Virtual Highest Priority Pending Interrupt Register 1 |
| AArch64 | ICV_IAR0_EL1 | Interrupt Controller Virtual Interrupt Acknowledge Register 0 |
| AArch64 | ICV_IAR1_EL1 | Interrupt Controller Virtual Interrupt Acknowledge Register 1 |
| AArch64 | ICV_IGRPEN0_EL1 | Interrupt Controller Virtual Interrupt Group 0 Enable register |
| AArch64 | ICV_IGRPEN1_EL1 | Interrupt Controller Virtual Interrupt Group 1 Enable register |
| AArch64 | ICV_NMIAR1_EL1 | Interrupt Controller Virtual Non-maskable Interrupt Acknowledge Register 1 |
| AArch64 | ICV_PMR_EL1 | Interrupt Controller Virtual Interrupt Priority Mask Register |
| AArch64 | ICV_RPR_EL1 | Interrupt Controller Virtual Running Priority Register |

In the GICD functional group:

| Exec state | Name | Description |
|------------|--|---|
| External | GICD_CLRSPI_NSR | Clear Non-secure SPI Pending Register |
| External | GICD_CLRSPI_SR | Clear Secure SPI Pending Register |
| External | GICD_CPENDSGIR<n> | SPI Clear-Pending Registers |
| External | GICD_CTLR | Distributor Control Register |
| External | GICD_ICACTIVER<n> | Interrupt Clear-Active Registers |
| External | GICD_ICACTIVER<n>E | Interrupt Clear-Active Registers (extended SPI range) |
| External | GICD_ICENABLER<n> | Interrupt Clear-Enable Registers |
| External | GICD_ICENABLER<n>E | Interrupt Clear-Enable Registers |
| External | GICD_ICFGR<n> | Interrupt Configuration Registers |
| External | GICD_ICFGR<n>E | Interrupt Configuration Registers (Extended SPI Range) |
| External | GICD_ICPENDR<n> | Interrupt Clear-Pending Registers |
| External | GICD_ICPENDR<n>E | Interrupt Clear-Pending Registers (extended SPI range) |
| External | GICD_IGROUPR<n> | Interrupt Group Registers |
| External | GICD_IGROUPR<n>E | Interrupt Group Registers (extended SPI range) |
| External | GICD_IGRPMODR<n> | Interrupt Group Modifier Registers |
| External | GICD_IGRPMODR<n>E | Interrupt Group Modifier Registers (extended SPI range) |
| External | GICD_IIDR | Distributor Implementer Identification Register |
| External | GICD_INMIR<n> | Non-maskable Interrupt Registers, x = 0 to 31 |
| External | GICD_INMIR<n>E | Non-maskable Interrupt Registers for Extended SPIs, x = 0 to 31 |

| Exec state | Name | Description |
|------------|---|---|
| External | GICD_IPRIORITYR<n> | Interrupt Priority Registers |
| External | GICD_IPRIORITYR<n>E | Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC. |
| External | GICD_IROUTER<n> | Interrupt Routing Registers |
| External | GICD_IROUTER<n>E | Interrupt Routing Registers (Extended SPI Range) |
| External | GICD_ISACTIVER<n> | Interrupt Set-Active Registers |
| External | GICD_ISACTIVER<n>E | Interrupt Set-Active Registers (extended SPI range) |
| External | GICD_ISENBALER<n> | Interrupt Set-Enable Registers |
| External | GICD_ISENBALER<n>E | Interrupt Set-Enable Registers |
| External | GICD_ISPENDR<n> | Interrupt Set-Pending Registers |
| External | GICD_ISPENDR<n>E | Interrupt Set-Pending Registers (extended SPI range) |
| External | GICD_ITARGETSR<n> | Interrupt Processor Targets Registers |
| External | GICD_NSACR<n> | Non-secure Access Control Registers |
| External | GICD_NSACR<n>E | Non-secure Access Control Registers |
| External | GICD_SETSPI_NSR | Set Non-secure SPI Pending Register |
| External | GICD_SETSPI_SR | Set Secure SPI Pending Register |
| External | GICD_SGIR | Software Generated Interrupt Register |
| External | GICD_SPENDSGIR<n> | SGI Set-Pending Registers |
| External | GICD_STATUSR | Error Reporting Status Register |
| External | GICD_TYPER | Interrupt Controller Type Register |
| External | GICD_TYPER2 | Interrupt Controller Type Register 2 |
| External | GICM_CLRSPI_NSR | Clear Non-secure SPI Pending Register |
| External | GICM_CLRSPI_SR | Clear Secure SPI Pending Register |
| External | GICM_IIDR | Distributor Implementer Identification Register |
| External | GICM_SETSPI_NSR | Set Non-secure SPI Pending Register |
| External | GICM_SETSPI_SR | Set Secure SPI Pending Register |
| External | GICM_TYPER | Distributor MSI Type Register |

In the GICR functional group:

| Exec state | Name | Description |
|------------|---|---|
| External | GICR_CLRLPIR | Clear LPI Pending Register |
| External | GICR_CTLR | Redistributor Control Register |
| External | GICR_ICACTIVER0 | Interrupt Clear-Active Register 0 |
| External | GICR_ICACTIVER<n>E | Interrupt Clear-Active Registers |
| External | GICR_ICENABLER0 | Interrupt Clear-Enable Register 0 |
| External | GICR_ICENABLER<n>E | Interrupt Clear-Enable Registers |
| External | GICR_ICFGR0 | Interrupt Configuration Register 0 |
| External | GICR_ICFGR1 | Interrupt Configuration Register 1 |
| External | GICR_ICFGR<n>E | Interrupt configuration registers |
| External | GICR_ICPENDR0 | Interrupt Clear-Pending Register 0 |
| External | GICR_ICPENDR<n>E | Interrupt Clear-Pending Registers |
| External | GICR_IGROUPR0 | Interrupt Group Register 0 |
| External | GICR_IGROUPR<n>E | Interrupt Group Registers |
| External | GICR_IGRPMODR0 | Interrupt Group Modifier Register 0 |
| External | GICR_IGRPMODR<n>E | Interrupt Group Modifier Registers |
| External | GICR_IIDR | Redistributor Implementer Identification Register |
| External | GICR_INMIR0 | Non-maskable Interrupt Register for PPIs. |
| External | GICR_INMIR<n>E | Non-maskable Interrupt Registers for Extended PPIs, x = 1 to 2. |
| External | GICR_INVALLR | Redistributor Invalidate All Register |
| External | GICR_INVLPIR | Redistributor Invalidate LPI Register |
| External | GICR_IPRIORITYR<n> | Interrupt Priority Registers |
| External | GICR_IPRIORITYR<n>E | Interrupt Priority Registers (extended PPI range) |
| External | GICR_ISACTIVER0 | Interrupt Set-Active Register 0 |
| External | GICR_ISACTIVER<n>E | Interrupt Set-Active Registers |
| External | GICR_ISENBALER0 | Interrupt Set-Enable Register 0 |
| External | GICR_ISENBALER<n>E | Interrupt Set-Enable Registers |
| External | GICR_ISPENDR0 | Interrupt Set-Pending Register 0 |
| External | GICR_ISPENDR<n>E | Interrupt Set-Pending Registers |
| External | GICR_MPAMIDR | Report maximum PARTID and PMG Register |
| External | GICR_NSACR | Non-secure Access Control Register |
| External | GICR_PARTIDR | Set PARTID and PMG Register |
| External | GICR_PENDBASER | Redistributor LPI Pending Table Base Address Register |

| Exec state | Name | Description |
|------------|---------------------------------|---|
| External | GICR_PROPBASER | Redistributor Properties Base Address Register |
| External | GICR_SETLPIR | Set LPI Pending Register |
| External | GICR_STATUSR | Error Reporting Status Register |
| External | GICR_SYNCR | Redistributor Synchronize Register |
| External | GICR_TYPER | Redistributor Type Register |
| External | GICR_VPENDBASER | Virtual Redistributor LPI Pending Table Base Address Register |
| External | GICR_VPROPBASER | Virtual Redistributor Properties Base Address Register |
| External | GICR_VSGIPENDR | Redistributor virtual SGI pending state register |
| External | GICR_VSGIR | Redistributor virtual SGI pending state request register |
| External | GICR_WAKER | Redistributor Wake Register |

In the GICC functional group:

| Exec state | Name | Description |
|------------|-------------------------------------|---|
| External | GICC_ABPR | CPU Interface Aliased Binary Point Register |
| External | GICC_AEOIR | CPU Interface Aliased End Of Interrupt Register |
| External | GICC_AHPPIR | CPU Interface Aliased Highest Priority Pending Interrupt Register |
| External | GICC_AIAR | CPU Interface Aliased Interrupt Acknowledge Register |
| External | GICC_APR<n> | CPU Interface Active Priorities Registers |
| External | GICC_BPR | CPU Interface Binary Point Register |
| External | GICC_CTLR | CPU Interface Control Register |
| External | GICC_DIR | CPU Interface Deactivate Interrupt Register |
| External | GICC_EOIR | CPU Interface End Of Interrupt Register |
| External | GICC_HPPIR | CPU Interface Highest Priority Pending Interrupt Register |
| External | GICC_IAR | CPU Interface Interrupt Acknowledge Register |
| External | GICC_IIDR | CPU Interface Identification Register |
| External | GICC_NSAPR<n> | CPU Interface Non-secure Active Priorities Registers |
| External | GICC_PMR | CPU Interface Priority Mask Register |
| External | GICC_RPR | CPU Interface Running Priority Register |
| External | GICC_STATUSR | CPU Interface Status Register |

In the GICV functional group:

| Exec state | Name | Description |
|------------|-----------------------------------|---|
| External | GICV_ABPR | Virtual Machine Aliased Binary Point Register |
| External | GICV_AEOIR | Virtual Machine Aliased End Of Interrupt Register |
| External | GICV_AHPPIR | Virtual Machine Aliased Highest Priority Pending Interrupt Register |
| External | GICV_AIAR | Virtual Machine Aliased Interrupt Acknowledge Register |
| External | GICV_APR<n> | Virtual Machine Active Priorities Registers |
| External | GICV_BPR | Virtual Machine Binary Point Register |
| External | GICV_CTLR | Virtual Machine Control Register |
| External | GICV_DIR | Virtual Machine Deactivate Interrupt Register |
| External | GICV_EOIR | Virtual Machine End Of Interrupt Register |
| External | GICV_HPPIR | Virtual Machine Highest Priority Pending Interrupt Register |
| External | GICV_IAR | Virtual Machine Interrupt Acknowledge Register |
| External | GICV_IIDR | Virtual Machine CPU Interface Identification Register |
| External | GICV_PMR | Virtual Machine Priority Mask Register |
| External | GICV_RPR | Virtual Machine Running Priority Register |
| External | GICV_STATUSR | Virtual Machine Error Reporting Status Register |

In the GICH functional group:

| Exec state | Name | Description |
|------------|-----------------------------------|---------------------------------------|
| External | GICH_APR<n> | Active Priorities Registers |
| External | GICH_EISR | End Interrupt Status Register |
| External | GICH_ELRSR | Empty List Register Status Register |
| External | GICH_HCR | Hypervisor Control Register |
| External | GICH_LR<n> | List Registers |
| External | GICH_MISR | Maintenance Interrupt Status Register |
| External | GICH_VMCR | Virtual Machine Control Register |
| External | GICH_VTR | Virtual Type Register |

In the GITS functional group:

| Exec state | Name | Description |
|------------|-------------------------------------|--|
| External | GITS_BASER<n> | ITS Translation Table Descriptors |
| External | GITS_CBASER | ITS Command Queue Descriptor |
| External | GITS_CREADR | ITS Read Register |
| External | GITS_CTLR | ITS Control Register |
| External | GITS_CWRITER | ITS Write Register |
| External | GITS_IIDR | ITS Identification Register |
| External | GITS_MPAMIDR | Report maximum PARTID and PMG Register |
| External | GITS_MPIDR | Report ITS's affinity. |
| External | GITS_PARTIDR | Set PARTID and PMG Register |
| External | GITS_SGIR | ITS SGI Register |
| External | GITS_STATUSR | ITS Error Reporting Status Register |
| External | GITS_TRANSLATER | ITS Translation Register |
| External | GITS_TYPER | ITS Type Register |
| External | GITS_UMSIR | ITS Unmapped MSI register |

In the RAS functional group:

| Exec state | Name | Description |
|------------|-----------------------------------|---|
| AArch32 | DISR | Deferred Interrupt Status Register |
| AArch32 | ERRIDR | Error Record ID Register |
| AArch32 | ERRSELR | Error Record Select Register |
| AArch32 | ERXADDR | Selected Error Record Address Register |
| AArch32 | ERXADDR2 | Selected Error Record Address Register 2 |
| AArch32 | ERXCTLR | Selected Error Record Control Register |
| AArch32 | ERXCTLR2 | Selected Error Record Control Register 2 |
| AArch32 | ERXFR | Selected Error Record Feature Register |
| AArch32 | ERXFR2 | Selected Error Record Feature Register 2 |
| AArch32 | ERXMISC0 | Selected Error Record Miscellaneous Register 0 |
| AArch32 | ERXMISC1 | Selected Error Record Miscellaneous Register 1 |
| AArch32 | ERXMISC2 | Selected Error Record Miscellaneous Register 2 |
| AArch32 | ERXMISC3 | Selected Error Record Miscellaneous Register 3 |
| AArch32 | ERXMISC4 | Selected Error Record Miscellaneous Register 4 |
| AArch32 | ERXMISC5 | Selected Error Record Miscellaneous Register 5 |
| AArch32 | ERXMISC6 | Selected Error Record Miscellaneous Register 6 |
| AArch32 | ERXMISC7 | Selected Error Record Miscellaneous Register 7 |
| AArch32 | ERXSTATUS | Selected Error Record Primary Status Register |
| AArch32 | VDFSR | Virtual SError Exception Syndrome Register |
| AArch32 | VDISR | Virtual Deferred Interrupt Status Register |
| AArch64 | DISR_EL1 | Deferred Interrupt Status Register |
| AArch64 | ERRIDR_EL1 | Error Record ID Register |
| AArch64 | ERRSELR_EL1 | Error Record Select Register |
| AArch64 | ERXADDR_EL1 | Selected Error Record Address Register |
| AArch64 | ERXCTLR_EL1 | Selected Error Record Control Register |
| AArch64 | ERXFR_EL1 | Selected Error Record Feature Register |
| AArch64 | ERXMISC0_EL1 | Selected Error Record Miscellaneous Register 0 |
| AArch64 | ERXMISC1_EL1 | Selected Error Record Miscellaneous Register 1 |
| AArch64 | ERXMISC2_EL1 | Selected Error Record Miscellaneous Register 2 |
| AArch64 | ERXMISC3_EL1 | Selected Error Record Miscellaneous Register 3 |
| AArch64 | ERXPFGCDN_EL1 | Selected Pseudo-fault Generation Countdown register |
| AArch64 | ERXPFGCTL_EL1 | Selected Pseudo-fault Generation Control register |
| AArch64 | ERXPFGF_EL1 | Selected Pseudo-fault Generation Feature register |
| AArch64 | ERXSTATUS_EL1 | Selected Error Record Primary Status Register |
| AArch64 | VDISR_EL2 | Virtual Deferred Interrupt Status Register |
| AArch64 | VSESR_EL2 | Virtual SError Exception Syndrome Register |
| External | ERR<n>ADDR | Error Record Address Register |
| External | ERR<n>CTLR | Error Record Control Register |
| External | ERR<n>FR | Error Record Feature Register |
| External | ERR<n>MISC0 | Error Record Miscellaneous Register 0 |
| External | ERR<n>MISC1 | Error Record Miscellaneous Register 1 |
| External | ERR<n>MISC2 | Error Record Miscellaneous Register 2 |
| External | ERR<n>MISC3 | Error Record Miscellaneous Register 3 |

| Exec state | Name | Description |
|------------|------------------------------------|---|
| External | ERR<n>PFGCDN | Pseudo-fault Generation Countdown Register |
| External | ERR<n>PFGCTL | Pseudo-fault Generation Control Register |
| External | ERR<n>PFGF | Pseudo-fault Generation Feature Register |
| External | ERR<n>STATUS | Error Record Primary Status Register |
| External | ERRCIDR0 | Component Identification Register 0 |
| External | ERRCIDR1 | Component Identification Register 1 |
| External | ERRCIDR2 | Component Identification Register 2 |
| External | ERRCIDR3 | Component Identification Register 3 |
| External | ERRCRICR0 | Critical Error Interrupt Configuration Register 0 |
| External | ERRCRICR1 | Critical Error Interrupt Configuration Register 1 |
| External | ERRCRICR2 | Critical Error Interrupt Configuration Register 2 |
| External | ERRDEVAFF | Device Affinity Register |
| External | ERRDEVARCH | Device Architecture Register |
| External | ERRDEVID | Device Configuration Register |
| External | ERRERICR0 | Error Recovery Interrupt Configuration Register 0 |
| External | ERRERICR1 | Error Recovery Interrupt Configuration Register 1 |
| External | ERRERICR2 | Error Recovery Interrupt Configuration Register 2 |
| External | ERRFHICR0 | Fault Handling Interrupt Configuration Register 0 |
| External | ERRFHICR1 | Fault Handling Interrupt Configuration Register 1 |
| External | ERRFHICR2 | Fault Handling Interrupt Configuration Register 2 |
| External | ERRGSR | Error Group Status Register |
| External | ERRIIDR | Implementation Identification Register |
| External | ERRIMPDEF<n> | IMPLEMENTATION DEFINED Register <n> |
| External | ERRIRQCR<n> | Generic Error Interrupt Configuration Register |
| External | ERRIRQSR | Error Interrupt Status Register |
| External | ERRPIDR0 | Peripheral Identification Register 0 |
| External | ERRPIDR1 | Peripheral Identification Register 1 |
| External | ERRPIDR2 | Peripheral Identification Register 2 |
| External | ERRPIDR3 | Peripheral Identification Register 3 |
| External | ERRPIDR4 | Peripheral Identification Register 4 |

In the MPAM functional group:

| Exec state | Name | Description |
|------------|--|---|
| AArch64 | MPAM0_EL1 | MPAM0 Register (EL1) |
| AArch64 | MPAM1_EL1 | MPAM1 Register (EL1) |
| AArch64 | MPAM2_EL2 | MPAM2 Register (EL2) |
| AArch64 | MPAM3_EL3 | MPAM3 Register (EL3) |
| AArch64 | MPAMHCR_EL2 | MPAM Hypervisor Control Register (EL2) |
| AArch64 | MPAMSM_EL1 | MPAM Streaming Mode Register |
| AArch64 | MPAMVPM0_EL2 | MPAM Virtual PARTID Mapping Register 0 |
| AArch64 | MPAMVPM1_EL2 | MPAM Virtual PARTID Mapping Register 1 |
| AArch64 | MPAMVPM2_EL2 | MPAM Virtual PARTID Mapping Register 2 |
| AArch64 | MPAMVPM3_EL2 | MPAM Virtual PARTID Mapping Register 3 |
| AArch64 | MPAMVPM4_EL2 | MPAM Virtual PARTID Mapping Register 4 |
| AArch64 | MPAMVPM5_EL2 | MPAM Virtual PARTID Mapping Register 5 |
| AArch64 | MPAMVPM6_EL2 | MPAM Virtual PARTID Mapping Register 6 |
| AArch64 | MPAMVPM7_EL2 | MPAM Virtual PARTID Mapping Register 7 |
| AArch64 | MPAMVPMV_EL2 | MPAM Virtual Partition Mapping Valid Register |
| External | MPAMCFG_CASSOC | MPAM Cache Maximum Associativity Partition Configuration Register |
| External | MPAMCFG_CMAX | MPAM Cache Maximum Capacity Partition Configuration Register |
| External | MPAMCFG_CMIN | MPAM Cache Minimum Capacity Partition Configuration Register |
| External | MPAMCFG_CPBM<n> | MPAM Cache Portion Bitmap Partition Configuration Register |
| External | MPAMCFG_DIS | MPAM Partition Configuration Disable Register |
| External | MPAMCFG_EN | MPAM Partition Configuration Enable Register |
| External | MPAMCFG_EN_FLAGS | MPAM Partition Configuration Enable Flags Register |
| External | MPAMCFG_INTPARTID | MPAM Internal PARTID Narrowing Configuration Register |
| External | MPAMCFG_MBW_MAX | MPAM Memory Bandwidth Maximum Partition Configuration Register |
| External | MPAMCFG_MBW_MIN | MPAM Memory Bandwidth Minimum Partition Configuration Register |
| External | MPAMCFG_MBW_PBM<n> | MPAM Bandwidth Portion Bitmap Partition Configuration Register |

| Exec state | Name | Description |
|------------|--|--|
| External | MPAMCFG_MBW_PROP | MPAM Memory Bandwidth Proportional Stride Partition Configuration Register |
| External | MPAMCFG_MBW_WINWD | MPAM Memory Bandwidth Partitioning Window Width Configuration Register |
| External | MPAMCFG_PART_SEL | MPAM Partition Configuration Selection Register |
| External | MPAMCFG_PRI | MPAM Priority Partition Configuration Register |
| External | MPAMF_AIDR | MPAM Architecture Identification Register |
| External | MPAMF_CCAP_IDR | MPAM Features Cache Capacity Partitioning ID register |
| External | MPAMF_CPOR_IDR | MPAM Features Cache Portion Partitioning ID register |
| External | MPAMF_CSUMON_IDR | MPAM Features Cache Storage Usage Monitoring ID register |
| External | MPAMF_ECR | MPAM Error Control Register |
| External | MPAMF_ERR_MSI_ADDR_H | MPAM Error MSI High-part Address Register |
| External | MPAMF_ERR_MSI_ADDR_L | MPAM Error MSI Low-part Address Register |
| External | MPAMF_ERR_MSI_ATTR | MPAM Error MSI Write Attributes Register |
| External | MPAMF_ERR_MSI_DATA | MPAM Error MSI Data Register |
| External | MPAMF_ERR_MSI_MPAM | MPAM Error MSI Write MPAM Information Register |
| External | MPAMF_ESR | MPAM Error Status Register |
| External | MPAMF_IDR | MPAM Features Identification Register |
| External | MPAMF_IIDR | MPAM Implementation Identification Register |
| External | MPAMF_IMPL_IDR | MPAM Implementation-Specific Partitioning Feature Identification Register |
| External | MPAMF_MBWUMON_IDR | MPAM Features Memory Bandwidth Usage Monitoring ID register |
| External | MPAMF_MBW_IDR | MPAM Memory Bandwidth Partitioning Identification Register |
| External | MPAMF_MSMON_IDR | MPAM Resource Monitoring Identification Register |
| External | MPAMF_PARTID_NRW_IDR | MPAM PARTID Narrowing ID register |
| External | MPAMF_PRI_IDR | MPAM Priority Partitioning Identification Register |
| External | MPAMF_SIDR | MPAM Features Secure Identification Register |
| External | MSMON_CAPT_EVNT | MPAM Capture Event Generation Register |
| External | MSMON_CFG_CSU_CTL | MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register |
| External | MSMON_CFG_CSU_FLT | MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register |
| External | MSMON_CFG_MBWU_CTL | MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register |
| External | MSMON_CFG_MBWU_FLT | MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register |
| External | MSMON_CFG_MON_SEL | MPAM Monitor Instance Selection Register |
| External | MSMON_CSU | MPAM Cache Storage Usage Monitor Register |
| External | MSMON_CSU_CAPTURE | MPAM Cache Storage Usage Monitor Capture Register |
| External | MSMON_CSU_OFSR | MPAM CSU Monitor Overflow Status Register |
| External | MSMON_MBWU | MPAM Memory Bandwidth Usage Monitor Register |
| External | MSMON_MBWU_CAPTURE | MPAM Memory Bandwidth Usage Monitor Capture Register |
| External | MSMON_MBWU_L | MPAM Long Memory Bandwidth Usage Monitor Register |
| External | MSMON_MBWU_L_CAPTURE | MPAM Long Memory Bandwidth Usage Monitor Capture Register |
| External | MSMON_MBWU_OFSR | MPAM MBWU Monitor Overflow Status Register |
| External | MSMON_OFLOW_MSI_ADDR_H | MPAM Monitor Overflow MSI Write High-part Address Register |
| External | MSMON_OFLOW_MSI_ADDR_L | MPAM Monitor Overflow MSI Low-part Address Register |
| External | MSMON_OFLOW_MSI_ATTR | MPAM Monitor Overflow MSI Write Attributes Register |
| External | MSMON_OFLOW_MSI_DATA | MPAM Monitor Overflow MSI Write Data Register |
| External | MSMON_OFLOW_MSI_MPAM | MPAM Monitor Overflow MSI Write MPAM Information Register |
| External | MSMON_OFLOW_SR | MPAM Monitor Overflow Status Register |

In the Pointer authentication functional group:

| Exec state | Name | Description |
|------------|-------------------------------|---|
| AArch64 | APDAKeyHi_EL1 | Pointer Authentication Key A for Data (bits[127:64]) |
| AArch64 | APDAKeyLo_EL1 | Pointer Authentication Key A for Data (bits[63:0]) |
| AArch64 | APDBKeyHi_EL1 | Pointer Authentication Key B for Data (bits[127:64]) |
| AArch64 | APDBKeyLo_EL1 | Pointer Authentication Key B for Data (bits[63:0]) |
| AArch64 | APGAKeyHi_EL1 | Pointer Authentication Key A for Code (bits[127:64]) |
| AArch64 | APGAKeyLo_EL1 | Pointer Authentication Key A for Code (bits[63:0]) |
| AArch64 | APIAKeyHi_EL1 | Pointer Authentication Key A for Instruction (bits[127:64]) |
| AArch64 | APIAKeyLo_EL1 | Pointer Authentication Key A for Instruction (bits[63:0]) |

| Exec state | Name | Description |
|------------|-------------------------------|---|
| AArch64 | APIBKeyHi_EL1 | Pointer Authentication Key B for Instruction (bits[127:64]) |
| AArch64 | APIBKeyLo_EL1 | Pointer Authentication Key B for Instruction (bits[63:0]) |

In the AMU functional group:

| Exec state | Name | Description |
|------------|---|--|
| AArch32 | AMCFGR | Activity Monitors Configuration Register |
| AArch32 | AMCGCR | Activity Monitors Counter Group Configuration Register |
| AArch32 | AMCNTENCLR0 | Activity Monitors Count Enable Clear Register 0 |
| AArch32 | AMCNTENCLR1 | Activity Monitors Count Enable Clear Register 1 |
| AArch32 | AMCNTENSET0 | Activity Monitors Count Enable Set Register 0 |
| AArch32 | AMCNTENSET1 | Activity Monitors Count Enable Set Register 1 |
| AArch32 | AMCR | Activity Monitors Control Register |
| AArch32 | AMEVCNTR0<n> | Activity Monitors Event Counter Registers 0 |
| AArch32 | AMEVCNTR1<n> | Activity Monitors Event Counter Registers 1 |
| AArch32 | AMEVTYPER0<n> | Activity Monitors Event Type Registers 0 |
| AArch32 | AMEVTYPER1<n> | Activity Monitors Event Type Registers 1 |
| AArch32 | AMUSERENR | Activity Monitors User Enable Register |
| AArch64 | AMCFGR_EL0 | Activity Monitors Configuration Register |
| AArch64 | AMCG1IDR_EL0 | Activity Monitors Counter Group 1 Identification Register |
| AArch64 | AMCGCR_EL0 | Activity Monitors Counter Group Configuration Register |
| AArch64 | AMCNTENCLR0_EL0 | Activity Monitors Count Enable Clear Register 0 |
| AArch64 | AMCNTENCLR1_EL0 | Activity Monitors Count Enable Clear Register 1 |
| AArch64 | AMCNTENSET0_EL0 | Activity Monitors Count Enable Set Register 0 |
| AArch64 | AMCNTENSET1_EL0 | Activity Monitors Count Enable Set Register 1 |
| AArch64 | AMCR_EL0 | Activity Monitors Control Register |
| AArch64 | AMEVCNTR0<n>_EL0 | Activity Monitors Event Counter Registers 0 |
| AArch64 | AMEVCNTR1<n>_EL0 | Activity Monitors Event Counter Registers 1 |
| AArch64 | AMEVCNTVOFF0<n>_EL2 | Activity Monitors Event Counter Virtual Offset Registers 0 |
| AArch64 | AMEVCNTVOFF1<n>_EL2 | Activity Monitors Event Counter Virtual Offset Registers 1 |
| AArch64 | AMEVTYPER0<n>_EL0 | Activity Monitors Event Type Registers 0 |
| AArch64 | AMEVTYPER1<n>_EL0 | Activity Monitors Event Type Registers 1 |
| AArch64 | AMUSERENR_EL0 | Activity Monitors User Enable Register |
| External | AMCFGR | Activity Monitors Configuration Register |
| External | AMCGCR | Activity Monitors Counter Group Configuration Register |
| External | AMCIDR0 | Activity Monitors Component Identification Register 0 |
| External | AMCIDR1 | Activity Monitors Component Identification Register 1 |
| External | AMCIDR2 | Activity Monitors Component Identification Register 2 |
| External | AMCIDR3 | Activity Monitors Component Identification Register 3 |
| External | AMCNTENCLR0 | Activity Monitors Count Enable Clear Register 0 |
| External | AMCNTENCLR1 | Activity Monitors Count Enable Clear Register 1 |
| External | AMCNTENSET0 | Activity Monitors Count Enable Set Register 0 |
| External | AMCNTENSET1 | Activity Monitors Count Enable Set Register 1 |
| External | AMCR | Activity Monitors Control Register |
| External | AMDEVAFF0 | Activity Monitors Device Affinity Register 0 |
| External | AMDEVAFF1 | Activity Monitors Device Affinity Register 1 |
| External | AMDEVARCH | Activity Monitors Device Architecture Register |
| External | AMDEVTYPE | Activity Monitors Device Type Register |
| External | AMEVCNTR0<n> | Activity Monitors Event Counter Registers 0 |
| External | AMEVCNTR1<n> | Activity Monitors Event Counter Registers 1 |
| External | AMEVTYPER0<n> | Activity Monitors Event Type Registers 0 |
| External | AMEVTYPER1<n> | Activity Monitors Event Type Registers 1 |
| External | AMIIDR | Activity Monitors Implementation Identification Register |
| External | AMPIDR0 | Activity Monitors Peripheral Identification Register 0 |
| External | AMPIDR1 | Activity Monitors Peripheral Identification Register 1 |
| External | AMPIDR2 | Activity Monitors Peripheral Identification Register 2 |
| External | AMPIDR3 | Activity Monitors Peripheral Identification Register 3 |
| External | AMPIDR4 | Activity Monitors Peripheral Identification Register 4 |

In the Root functional group:

| Exec state | Name | Description |
|------------|---------------------------|---|
| AArch64 | GPCCR_EL3 | Granule Protection Check Control Register (EL3) |
| AArch64 | GPTBR_EL3 | Granule Protection Table Base Register |
| AArch64 | MFAR_EL3 | PA Fault Address Register |

In the GIC ITS registers functional group:

| Exec state | Name | Description |
|------------|-------------------------------------|--|
| External | GITS_BASER<n> | ITS Translation Table Descriptors |
| External | GITS_CBASER | ITS Command Queue Descriptor |
| External | GITS_CREADR | ITS Read Register |
| External | GITS_CTLR | ITS Control Register |
| External | GITS_CWRITER | ITS Write Register |
| External | GITS_IIDR | ITS Identification Register |
| External | GITS_MPAMIDR | Report maximum PARTID and PMG Register |
| External | GITS_MPIDR | Report ITS's affinity. |
| External | GITS_PARTIDR | Set PARTID and PMG Register |
| External | GITS_SGIR | ITS SGI Register |
| External | GITS_STATUSR | ITS Error Reporting Status Register |
| External | GITS_TRANSLATER | ITS Translation Register |
| External | GITS_TYPER | ITS Type Register |
| External | GITS_UMSIR | ITS Unmapped MSI register |

30/09/2021 14:57

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

External registers

[AMCFGR](#): Activity Monitors Configuration Register
[AMCGCR](#): Activity Monitors Counter Group Configuration Register
[AMCIDR0](#): Activity Monitors Component Identification Register 0
[AMCIDR1](#): Activity Monitors Component Identification Register 1
[AMCIDR2](#): Activity Monitors Component Identification Register 2
[AMCIDR3](#): Activity Monitors Component Identification Register 3
[AMCNTENCLR0](#): Activity Monitors Count Enable Clear Register 0
[AMCNTENCLR1](#): Activity Monitors Count Enable Clear Register 1
[AMCNTENSET0](#): Activity Monitors Count Enable Set Register 0
[AMCNTENSET1](#): Activity Monitors Count Enable Set Register 1
[AMCR](#): Activity Monitors Control Register
[AMDEVAFF0](#): Activity Monitors Device Affinity Register 0
[AMDEVAFF1](#): Activity Monitors Device Affinity Register 1
[AMDEVARCH](#): Activity Monitors Device Architecture Register
[AMDEVTYPE](#): Activity Monitors Device Type Register
[AMEVCNTR0<n>](#): Activity Monitors Event Counter Registers 0
[AMEVCNTR1<n>](#): Activity Monitors Event Counter Registers 1
[AMEVTYPER0<n>](#): Activity Monitors Event Type Registers 0
[AMEVTYPER1<n>](#): Activity Monitors Event Type Registers 1
[AMIIDR](#): Activity Monitors Implementation Identification Register
[AMPIDR0](#): Activity Monitors Peripheral Identification Register 0
[AMPIDR1](#): Activity Monitors Peripheral Identification Register 1
[AMPIDR2](#): Activity Monitors Peripheral Identification Register 2
[AMPIDR3](#): Activity Monitors Peripheral Identification Register 3
[AMPIDR4](#): Activity Monitors Peripheral Identification Register 4
[ASICCTL](#): CTI External Multiplexer Control register
[CNTACR<n>](#): Counter-timer Access Control Registers
[CNTCR](#): Counter Control Register
[CNTCV](#): Counter Count Value register
[CNTEL0ACR](#): Counter-timer EL0 Access Control Register
[CNTFID0](#): Counter Frequency ID
[CNTFID<n>](#): Counter Frequency IDs, $n > 0$
[CNTFRQ](#): Counter-timer Frequency
[CNTID](#): Counter Identification Register

[CNTNSAR](#): Counter-timer Non-secure Access Register

[CNTPCT](#): Counter-timer Physical Count

[CNTP_CTL](#): Counter-timer Physical Timer Control

[CNTP_CVAL](#): Counter-timer Physical Timer CompareValue

[CNTP_TVAL](#): Counter-timer Physical Timer TimerValue

[CNTSCR](#): Counter Scale Register

[CNTSR](#): Counter Status Register

[CNTTIDR](#): Counter-timer Timer ID Register

[CNTVCT](#): Counter-timer Virtual Count

[CNTVOFF](#): Counter-timer Virtual Offset

[CNTVOFF<n>](#): Counter-timer Virtual Offsets

[CNTV_CTL](#): Counter-timer Virtual Timer Control

[CNTV_CVAL](#): Counter-timer Virtual Timer CompareValue

[CNTV_TVAL](#): Counter-timer Virtual Timer TimerValue

[CounterID<n>](#): Counter ID registers

[CTIAPPCLEAR](#): CTI Application Trigger Clear register

[CTIAPPULSE](#): CTI Application Pulse register

[CTIAPPSET](#): CTI Application Trigger Set register

[CTIAUTHSTATUS](#): CTI Authentication Status register

[CTICHINSTATUS](#): CTI Channel In Status register

[CTICHOUTSTATUS](#): CTI Channel Out Status register

[CTICIDR0](#): CTI Component Identification Register 0

[CTICIDR1](#): CTI Component Identification Register 1

[CTICIDR2](#): CTI Component Identification Register 2

[CTICIDR3](#): CTI Component Identification Register 3

[CTICLAIMCLR](#): CTI CLAIM Tag Clear register

[CTICLAIMSET](#): CTI CLAIM Tag Set register

[CTICONTROL](#): CTI Control register

[CTIDEVAFF0](#): CTI Device Affinity register 0

[CTIDEVAFF1](#): CTI Device Affinity register 1

[CTIDEVARCH](#): CTI Device Architecture register

[CTIDEVCTL](#): CTI Device Control register

[CTIDEVID](#): CTI Device ID register 0

[CTIDEVID1](#): CTI Device ID register 1

[CTIDEVID2](#): CTI Device ID register 2

[CTIDEVTYPE](#): CTI Device Type register

[CTIGATE](#): CTI Channel Gate Enable register

[CTIINEN<n>](#): CTI Input Trigger to Output Channel Enable registers

[CTIINTACK](#): CTI Output Trigger Acknowledge register

[CTIITCTRL](#): CTI Integration mode Control register

[CTILAR](#): CTI Lock Access Register

[CTILSR](#): CTI Lock Status Register

[CTIOUTEN<n>](#): CTI Input Channel to Output Trigger Enable registers

[CTIPIDR0](#): CTI Peripheral Identification Register 0

[CTIPIDR1](#): CTI Peripheral Identification Register 1

[CTIPIDR2](#): CTI Peripheral Identification Register 2

[CTIPIDR3](#): CTI Peripheral Identification Register 3

[CTIPIDR4](#): CTI Peripheral Identification Register 4

[CTITRIGINSTATUS](#): CTI Trigger In Status register

[CTITRIGOUTSTATUS](#): CTI Trigger Out Status register

[DBGAUTHSTATUS_EL1](#): Debug Authentication Status register

[DBGBCR<n>_EL1](#): Debug Breakpoint Control Registers

[DBGBVR<n>_EL1](#): Debug Breakpoint Value Registers

[DBGCLAIMCLR_EL1](#): Debug CLAIM Tag Clear register

[DBGCLAIMSET_EL1](#): Debug CLAIM Tag Set register

[DBGDTRRX_EL0](#): Debug Data Transfer Register, Receive

[DBGDTRTX_EL0](#): Debug Data Transfer Register, Transmit

[DBGWCR<n>_EL1](#): Debug Watchpoint Control Registers

[DBGWVR<n>_EL1](#): Debug Watchpoint Value Registers

[EDAA32PFR](#): External Debug Auxiliary Processor Feature Register

[EDACR](#): External Debug Auxiliary Control Register

[EDCIDR0](#): External Debug Component Identification Register 0

[EDCIDR1](#): External Debug Component Identification Register 1

[EDCIDR2](#): External Debug Component Identification Register 2

[EDCIDR3](#): External Debug Component Identification Register 3

[EDCIDSr](#): External Debug Context ID Sample Register

[EDDEVAFF0](#): External Debug Device Affinity register 0

[EDDEVAFF1](#): External Debug Device Affinity register 1

[EDDEVARCH](#): External Debug Device Architecture register

[EDDEVID](#): External Debug Device ID register 0

[EDDEVID1](#): External Debug Device ID register 1

[EDDEVID2](#): External Debug Device ID register 2

[EDDEVTYPE](#): External Debug Device Type register

[EDDFR](#): External Debug Feature Register

[EDECCR](#): External Debug Exception Catch Control Register

[EDECR](#): External Debug Execution Control Register

[EDES](#): External Debug Event Status Register

[EDITCTRL](#): External Debug Integration mode Control register

[EDITR](#): External Debug Instruction Transfer Register

[EDLAR](#): External Debug Lock Access Register

[EDLSR](#): External Debug Lock Status Register

[EDPCSR](#): External Debug Program Counter Sample Register

[EDPFR](#): External Debug Processor Feature Register

[EDPIDR0](#): External Debug Peripheral Identification Register 0

[EDPIDR1](#): External Debug Peripheral Identification Register 1

[EDPIDR2](#): External Debug Peripheral Identification Register 2

[EDPIDR3](#): External Debug Peripheral Identification Register 3

[EDPIDR4](#): External Debug Peripheral Identification Register 4

[EDPRCR](#): External Debug Power/Reset Control Register

[EDPRSR](#): External Debug Processor Status Register

[EDRCR](#): External Debug Reserve Control Register

[EDSCR](#): External Debug Status and Control Register

[EDVIDSR](#): External Debug Virtual Context Sample Register

[EDWAR](#): External Debug Watchpoint Address Register

[ERR<n>ADDR](#): Error Record Address Register

[ERR<n>CTLR](#): Error Record Control Register

[ERR<n>FR](#): Error Record Feature Register

[ERR<n>MISC0](#): Error Record Miscellaneous Register 0

[ERR<n>MISC1](#): Error Record Miscellaneous Register 1

[ERR<n>MISC2](#): Error Record Miscellaneous Register 2

[ERR<n>MISC3](#): Error Record Miscellaneous Register 3

[ERR<n>PFGCDN](#): Pseudo-fault Generation Countdown Register

[ERR<n>PFGCTL](#): Pseudo-fault Generation Control Register

[ERR<n>PFGF](#): Pseudo-fault Generation Feature Register

[ERR<n>STATUS](#): Error Record Primary Status Register

[ERRCIDR0](#): Component Identification Register 0

[ERRCIDR1](#): Component Identification Register 1

[ERRCIDR2](#): Component Identification Register 2

[ERRCIDR3](#): Component Identification Register 3

[ERRCRICR0](#): Critical Error Interrupt Configuration Register 0

[ERRCRICR1](#): Critical Error Interrupt Configuration Register 1

[ERRCRICR2](#): Critical Error Interrupt Configuration Register 2

[ERRDEVAFF](#): Device Affinity Register

[ERRDEVARCH](#): Device Architecture Register

[ERRDEVID](#): Device Configuration Register

[ERRERICR0](#): Error Recovery Interrupt Configuration Register 0

[ERRERICR1](#): Error Recovery Interrupt Configuration Register 1

[ERRERICR2](#): Error Recovery Interrupt Configuration Register 2

[ERRFHICR0](#): Fault Handling Interrupt Configuration Register 0

[ERRFHICR1](#): Fault Handling Interrupt Configuration Register 1

[ERRFHICR2](#): Fault Handling Interrupt Configuration Register 2

[ERRGSR](#): Error Group Status Register

[ERRIIDR](#): Implementation Identification Register

[ERRIMPDEF<n>](#): IMPLEMENTATION DEFINED Register <n>

[ERRIRQCR<n>](#): Generic Error Interrupt Configuration Register

[ERRIRQSR](#): Error Interrupt Status Register

[ERRPIDR0](#): Peripheral Identification Register 0

[ERRPIDR1](#): Peripheral Identification Register 1

[ERRPIDR2](#): Peripheral Identification Register 2

[ERRPIDR3](#): Peripheral Identification Register 3

[ERRPIDR4](#): Peripheral Identification Register 4

[GICC_ABPR](#): CPU Interface Aliased Binary Point Register

[GICC_AEOIR](#): CPU Interface Aliased End Of Interrupt Register

[GICC_AHPPIR](#): CPU Interface Aliased Highest Priority Pending Interrupt Register

[GICC_AIAR](#): CPU Interface Aliased Interrupt Acknowledge Register

[GICC_APR<n>](#): CPU Interface Active Priorities Registers

[GICC_BPR](#): CPU Interface Binary Point Register

[GICC_CTLR](#): CPU Interface Control Register

[GICC_DIR](#): CPU Interface Deactivate Interrupt Register

[GICC_EOIR](#): CPU Interface End Of Interrupt Register

[GICC_HPPIR](#): CPU Interface Highest Priority Pending Interrupt Register

[GICC_IAR](#): CPU Interface Interrupt Acknowledge Register

[GICC_IIDR](#): CPU Interface Identification Register

[GICC_NSAPR<n>](#): CPU Interface Non-secure Active Priorities Registers

[GICC_PMR](#): CPU Interface Priority Mask Register

[GICC_RPR](#): CPU Interface Running Priority Register

[GICC_STATUSR](#): CPU Interface Status Register

[GICD_CLRSPI_NSR](#): Clear Non-secure SPI Pending Register

[GICD_CLRSPI_SR](#): Clear Secure SPI Pending Register

[GICD_CPENDSGIR<n>](#): SGI Clear-Pending Registers

[GICD_CTLR](#): Distributor Control Register

[GICD_ICACTIVER<n>](#): Interrupt Clear-Active Registers

[GICD_ICACTIVER<n>E](#): Interrupt Clear-Active Registers (extended SPI range)

[GICD_ICENABLER<n>](#): Interrupt Clear-Enable Registers

[GICD_ICENABLER<n>E](#): Interrupt Clear-Enable Registers

[GICD_ICFGR<n>](#): Interrupt Configuration Registers

[GICD_ICFGR<n>E](#): Interrupt Configuration Registers (Extended SPI Range)

[GICD_ICPENDR<n>](#): Interrupt Clear-Pending Registers

[GICD_ICPENDR<n>E](#): Interrupt Clear-Pending Registers (extended SPI range)

[GICD_IGROUPR<n>](#): Interrupt Group Registers

[GICD_IGROUPR<n>E](#): Interrupt Group Registers (extended SPI range)

[GICD_IGRPMODR<n>](#): Interrupt Group Modifier Registers

[GICD_IGRPMODR<n>E](#): Interrupt Group Modifier Registers (extended SPI range)

[GICD_IIDR](#): Distributor Implementer Identification Register

[GICD_INMIR<n>](#): Non-maskable Interrupt Registers, x = 0 to 31

[GICD_INMIR<n>E](#): Non-maskable Interrupt Registers for Extended SPIs, x = 0 to 31

[GICD_IPRIORITYR<n>](#): Interrupt Priority Registers

[GICD_IPRIORITYR<n>E](#): Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.

[GICD_IROUTER<n>](#): Interrupt Routing Registers

[GICD_IROUTER<n>E](#): Interrupt Routing Registers (Extended SPI Range)

[GICD_ISACTIVER<n>](#): Interrupt Set-Active Registers

[GICD_ISACTIVER<n>E](#): Interrupt Set-Active Registers (extended SPI range)

[GICD_ISENABLER<n>](#): Interrupt Set-Enable Registers

[GICD_ISENABLER<n>E](#): Interrupt Set-Enable Registers

[GICD_ISPENDR<n>](#): Interrupt Set-Pending Registers

[GICD_ISPENDR<n>E](#): Interrupt Set-Pending Registers (extended SPI range)

[GICD_ITARGETSR<n>](#): Interrupt Processor Targets Registers

[GICD_NSACR<n>](#): Non-secure Access Control Registers

[GICD_NSACR<n>E](#): Non-secure Access Control Registers

[GICD_SETSPI_NSR](#): Set Non-secure SPI Pending Register

[GICD_SETSPI_SR](#): Set Secure SPI Pending Register
[GICD_SGIR](#): Software Generated Interrupt Register
[GICD_SPENDSGIR<n>](#): SGI Set-Pending Registers
[GICD_STATUSR](#): Error Reporting Status Register
[GICD_TYPER](#): Interrupt Controller Type Register
[GICD_TYPER2](#): Interrupt Controller Type Register 2
[GICH_APR<n>](#): Active Priorities Registers
[GICH_EISR](#): End Interrupt Status Register
[GICH_ELRSR](#): Empty List Register Status Register
[GICH_HCR](#): Hypervisor Control Register
[GICH_LR<n>](#): List Registers
[GICH_MISR](#): Maintenance Interrupt Status Register
[GICH_VMCR](#): Virtual Machine Control Register
[GICH_VTR](#): Virtual Type Register
[GICM_CLRSPI_NSR](#): Clear Non-secure SPI Pending Register
[GICM_CLRSPI_SR](#): Clear Secure SPI Pending Register
[GICM_IIDR](#): Distributor Implementer Identification Register
[GICM_SETSPI_NSR](#): Set Non-secure SPI Pending Register
[GICM_SETSPI_SR](#): Set Secure SPI Pending Register
[GICM_TYPER](#): Distributor MSI Type Register
[GICR_CLRLPIR](#): Clear LPI Pending Register
[GICR_CTLR](#): Redistributor Control Register
[GICR_ICACTIVER0](#): Interrupt Clear-Active Register 0
[GICR_ICACTIVER<n>E](#): Interrupt Clear-Active Registers
[GICR_ICENABLER0](#): Interrupt Clear-Enable Register 0
[GICR_ICENABLER<n>E](#): Interrupt Clear-Enable Registers
[GICR_ICFGR0](#): Interrupt Configuration Register 0
[GICR_ICFGR1](#): Interrupt Configuration Register 1
[GICR_ICFGR<n>E](#): Interrupt configuration registers
[GICR_ICPENDR0](#): Interrupt Clear-Pending Register 0
[GICR_ICPENDR<n>E](#): Interrupt Clear-Pending Registers
[GICR_IGROUPR0](#): Interrupt Group Register 0
[GICR_IGROUPR<n>E](#): Interrupt Group Registers
[GICR_IGRPMODR0](#): Interrupt Group Modifier Register 0
[GICR_IGRPMODR<n>E](#): Interrupt Group Modifier Registers
[GICR_IIDR](#): Redistributor Implementer Identification Register

[GICR_INMIR0](#): Non-maskable Interrupt Register for PPIs.

[GICR_INMIR<n>E](#): Non-maskable Interrupt Registers for Extended PPIs, x = 1 to 2.

[GICR_INVALLR](#): Redistributor Invalidate All Register

[GICR_INVLPIR](#): Redistributor Invalidate LPI Register

[GICR_IPRIORITYR<n>](#): Interrupt Priority Registers

[GICR_IPRIORITYR<n>E](#): Interrupt Priority Registers (extended PPI range)

[GICR_ISACTIVER0](#): Interrupt Set-Active Register 0

[GICR_ISACTIVER<n>E](#): Interrupt Set-Active Registers

[GICR_ISENBALER0](#): Interrupt Set-Enable Register 0

[GICR_ISENBALER<n>E](#): Interrupt Set-Enable Registers

[GICR_ISPENDR0](#): Interrupt Set-Pending Register 0

[GICR_ISPENDR<n>E](#): Interrupt Set-Pending Registers

[GICR_MPAMIDR](#): Report maximum PARTID and PMG Register

[GICR_NSACR](#): Non-secure Access Control Register

[GICR_PARTIDR](#): Set PARTID and PMG Register

[GICR_PENDBASER](#): Redistributor LPI Pending Table Base Address Register

[GICR_PROPBASER](#): Redistributor Properties Base Address Register

[GICR_SETLPIR](#): Set LPI Pending Register

[GICR_STATUSR](#): Error Reporting Status Register

[GICR_SYNCR](#): Redistributor Synchronize Register

[GICR_TYPER](#): Redistributor Type Register

[GICR_VPENDBASER](#): Virtual Redistributor LPI Pending Table Base Address Register

[GICR_VPROPBASER](#): Virtual Redistributor Properties Base Address Register

[GICR_VSGIPENDR](#): Redistributor virtual SGI pending state register

[GICR_VSGIR](#): Redistributor virtual SGI pending state request register

[GICR_WAKER](#): Redistributor Wake Register

[GICV_ABPR](#): Virtual Machine Aliased Binary Point Register

[GICV_AEOIR](#): Virtual Machine Aliased End Of Interrupt Register

[GICV_AHPPIR](#): Virtual Machine Aliased Highest Priority Pending Interrupt Register

[GICV_AIAR](#): Virtual Machine Aliased Interrupt Acknowledge Register

[GICV_APR<n>](#): Virtual Machine Active Priorities Registers

[GICV_BPR](#): Virtual Machine Binary Point Register

[GICV_CTLR](#): Virtual Machine Control Register

[GICV_DIR](#): Virtual Machine Deactivate Interrupt Register

[GICV_EOIR](#): Virtual Machine End Of Interrupt Register

[GICV_HPPIR](#): Virtual Machine Highest Priority Pending Interrupt Register

[GICV_IAR](#): Virtual Machine Interrupt Acknowledge Register

[GICV_IIDR](#): Virtual Machine CPU Interface Identification Register

[GICV_PMR](#): Virtual Machine Priority Mask Register

[GICV_RPR](#): Virtual Machine Running Priority Register

[GICV_STATUSR](#): Virtual Machine Error Reporting Status Register

[GITS_BASER<n>](#): ITS Translation Table Descriptors

[GITS_CBASER](#): ITS Command Queue Descriptor

[GITS_CREADR](#): ITS Read Register

[GITS_CTLR](#): ITS Control Register

[GITS_CWRITER](#): ITS Write Register

[GITS_IIDR](#): ITS Identification Register

[GITS_MPAMIDR](#): Report maximum PARTID and PMG Register

[GITS_MPIDR](#): Report ITS's affinity.

[GITS_PARTIDR](#): Set PARTID and PMG Register

[GITS_SGIR](#): ITS SGI Register

[GITS_STATUSR](#): ITS Error Reporting Status Register

[GITS_TRANSLATER](#): ITS Translation Register

[GITS_TYPER](#): ITS Type Register

[GITS_UMSIR](#): ITS Unmapped MSI register

[MIDR_EL1](#): Main ID Register

[MPAMCFG_CASSOC](#): MPAM Cache Maximum Associativity Partition Configuration Register

[MPAMCFG_CMAX](#): MPAM Cache Maximum Capacity Partition Configuration Register

[MPAMCFG_CMIN](#): MPAM Cache Minimum Capacity Partition Configuration Register

[MPAMCFG_CPBM<n>](#): MPAM Cache Portion Bitmap Partition Configuration Register

[MPAMCFG_DIS](#): MPAM Partition Configuration Disable Register

[MPAMCFG_EN](#): MPAM Partition Configuration Enable Register

[MPAMCFG_EN_FLAGS](#): MPAM Partition Configuration Enable Flags Register

[MPAMCFG_INTPARTID](#): MPAM Internal PARTID Narrowing Configuration Register

[MPAMCFG_MBW_MAX](#): MPAM Memory Bandwidth Maximum Partition Configuration Register

[MPAMCFG_MBW_MIN](#): MPAM Memory Bandwidth Minimum Partition Configuration Register

[MPAMCFG_MBW_PBM<n>](#): MPAM Bandwidth Portion Bitmap Partition Configuration Register

[MPAMCFG_MBW_PROP](#): MPAM Memory Bandwidth Proportional Stride Partition Configuration Register

[MPAMCFG_MBW_WINWD](#): MPAM Memory Bandwidth Partitioning Window Width Configuration Register

[MPAMCFG_PART_SEL](#): MPAM Partition Configuration Selection Register

[MPAMCFG_PRI](#): MPAM Priority Partition Configuration Register

[MPAMF_AIDR](#): MPAM Architecture Identification Register

[MPAMF_CCAP_IDR](#): MPAM Features Cache Capacity Partitioning ID register

[MPAMF_CPOR_IDR](#): MPAM Features Cache Portion Partitioning ID register

[MPAMF_CSUMON_IDR](#): MPAM Features Cache Storage Usage Monitoring ID register

[MPAMF_ECR](#): MPAM Error Control Register

[MPAMF_ERR_MSI_ADDR_H](#): MPAM Error MSI High-part Address Register

[MPAMF_ERR_MSI_ADDR_L](#): MPAM Error MSI Low-part Address Register

[MPAMF_ERR_MSI_ATTR](#): MPAM Error MSI Write Attributes Register

[MPAMF_ERR_MSI_DATA](#): MPAM Error MSI Data Register

[MPAMF_ERR_MSI_MPAM](#): MPAM Error MSI Write MPAM Information Register

[MPAMF_ESR](#): MPAM Error Status Register

[MPAMF_IDR](#): MPAM Features Identification Register

[MPAMF_IIDR](#): MPAM Implementation Identification Register

[MPAMF_IMPL_IDR](#): MPAM Implementation-Specific Partitioning Feature Identification Register

[MPAMF_MBWUMON_IDR](#): MPAM Features Memory Bandwidth Usage Monitoring ID register

[MPAMF_MBW_IDR](#): MPAM Memory Bandwidth Partitioning Identification Register

[MPAMF_MSMON_IDR](#): MPAM Resource Monitoring Identification Register

[MPAMF_PARTID_NRW_IDR](#): MPAM PARTID Narrowing ID register

[MPAMF_PRI_IDR](#): MPAM Priority Partitioning Identification Register

[MPAMF_SIDR](#): MPAM Features Secure Identification Register

[MSMON_CAPT_EVT](#): MPAM Capture Event Generation Register

[MSMON_CFG_CSU_CTL](#): MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register

[MSMON_CFG_CSU_FLT](#): MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register

[MSMON_CFG_MBWU_CTL](#): MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register

[MSMON_CFG_MBWU_FLT](#): MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register

[MSMON_CFG_MON_SEL](#): MPAM Monitor Instance Selection Register

[MSMON_CSU](#): MPAM Cache Storage Usage Monitor Register

[MSMON_CSU_CAPTURE](#): MPAM Cache Storage Usage Monitor Capture Register

[MSMON_CSU_OFSR](#): MPAM CSU Monitor Overflow Status Register

[MSMON_MBWU](#): MPAM Memory Bandwidth Usage Monitor Register

[MSMON_MBWU_CAPTURE](#): MPAM Memory Bandwidth Usage Monitor Capture Register

[MSMON_MBWU_L](#): MPAM Long Memory Bandwidth Usage Monitor Register

[MSMON_MBWU_L_CAPTURE](#): MPAM Long Memory Bandwidth Usage Monitor Capture Register

[MSMON_MBWU_OFSR](#): MPAM MBWU Monitor Overflow Status Register

[MSMON_OFLOW_MSI_ADDR_H](#): MPAM Monitor Overflow MSI Write High-part Address Register

[MSMON_OFLOW_MSI_ADDR_L](#): MPAM Monitor Overflow MSI Low-part Address Register

[MSMON_OFLOW_MSI_ATTR](#): MPAM Monitor Overflow MSI Write Attributes Register

[MSMON_OFLOW_MSI_DATA](#): MPAM Monitor Overflow MSI Write Data Register

[MSMON_OFLOW_MSI_MPAM](#): MPAM Monitor Overflow MSI Write MPAM Information Register

[MSMON_OFLOW_SR](#): MPAM Monitor Overflow Status Register

[OSLAR_EL1](#): OS Lock Access Register

[PMAUTHSTATUS](#): Performance Monitors Authentication Status register

[PMCCFILTR_EL0](#): Performance Monitors Cycle Counter Filter Register

[PMCCNTR_EL0](#): Performance Monitors Cycle Counter

[PMCEID0](#): Performance Monitors Common Event Identification register 0

[PMCEID1](#): Performance Monitors Common Event Identification register 1

[PMCEID2](#): Performance Monitors Common Event Identification register 2

[PMCEID3](#): Performance Monitors Common Event Identification register 3

[PMCFGR](#): Performance Monitors Configuration Register

[PMCID1SR](#): CONTEXTIDR_EL1 Sample Register

[PMCID2SR](#): CONTEXTIDR_EL2 Sample Register

[PMCIDR0](#): Performance Monitors Component Identification Register 0

[PMCIDR1](#): Performance Monitors Component Identification Register 1

[PMCIDR2](#): Performance Monitors Component Identification Register 2

[PMCIDR3](#): Performance Monitors Component Identification Register 3

[PMCNTENCLR_EL0](#): Performance Monitors Count Enable Clear register

[PMCNTENSET_EL0](#): Performance Monitors Count Enable Set register

[PMCR_EL0](#): Performance Monitors Control Register

[PMDEVAFF0](#): Performance Monitors Device Affinity register 0

[PMDEVAFF1](#): Performance Monitors Device Affinity register 1

[PMDEVARCH](#): Performance Monitors Device Architecture register

[PMDEVID](#): Performance Monitors Device ID register

[PMDEVTYPE](#): Performance Monitors Device Type register

[PMEVCNTR<n>_EL0](#): Performance Monitors Event Count Registers

[PMEVFILTR<n>](#): Performance Monitors Event Type Select Register <n>

[PMEVTYPER<n>_EL0](#): Performance Monitors Event Type Registers

[PMINTENCLR_EL1](#): Performance Monitors Interrupt Enable Clear register

[PMINTENSET_EL1](#): Performance Monitors Interrupt Enable Set register

[PMITCTRL](#): Performance Monitors Integration mode Control register

[PMLAR](#): Performance Monitors Lock Access Register

[PMLSR](#): Performance Monitors Lock Status Register

[PMMIR](#): Performance Monitors Machine Identification Register

[PMOVSCLR_EL0](#): Performance Monitors Overflow Flag Status Clear register

[PMOVSSET_EL0](#): Performance Monitors Overflow Flag Status Set register

[PMPCSR](#): Program Counter Sample Register

[PMPIDR0](#): Performance Monitors Peripheral Identification Register 0

[PMPIDR1](#): Performance Monitors Peripheral Identification Register 1

[PMPIDR2](#): Performance Monitors Peripheral Identification Register 2

[PMPIDR3](#): Performance Monitors Peripheral Identification Register 3

[PMPIDR4](#): Performance Monitors Peripheral Identification Register 4

[PMSWINC_EL0](#): Performance Monitors Software Increment register

[PMVIDSR](#): VMID Sample Register

[TRCACATR<n>](#): Address Comparator Access Type Register <n>

[TRCACVR<n>](#): Address Comparator Value Register <n>

[TRCAUTHSTATUS](#): Authentication Status Register

[TRCAUXCTLR](#): Auxiliary Control Register

[TRCBBCTLR](#): Branch Broadcast Control Register

[TRCCCCTLR](#): Cycle Count Control Register

[TRCCIDCCTLR0](#): Context Identifier Comparator Control Register 0

[TRCCIDCCTLR1](#): Context Identifier Comparator Control Register 1

[TRCCIDCVR<n>](#): Context Identifier Comparator Value Registers <n>

[TRCCIDR0](#): Component Identification Register 0

[TRCCIDR1](#): Component Identification Register 1

[TRCCIDR2](#): Component Identification Register 2

[TRCCIDR3](#): Component Identification Register 3

[TRCCLAIMCLR](#): Claim Tag Clear Register

[TRCCLAIMSET](#): Claim Tag Set Register

[TRCCNTCTLR<n>](#): Counter Control Register <n>

[TRCCNTRLDVR<n>](#): Counter Reload Value Register <n>

[TRCCNTVR<n>](#): Counter Value Register <n>

[TRCCONFIGR](#): Trace Configuration Register

[TRCDEVAFF](#): Device Affinity Register

[TRCDEVARCH](#): Device Architecture Register

[TRCDEVID](#): Device Configuration Register

[TRCDEVID1](#): Device Configuration Register 1

[TRCDEVID2](#): Device Configuration Register 2

[TRCDEVTYPE](#): Device Type Register

[TRCEVENTCTL0R](#): Event Control 0 Register

[TRCEVENTCTL1R](#): Event Control 1 Register

[TRCEXTINSELR<n>](#): External Input Select Register <n>

[TRCIDR0](#): ID Register 0

[TRCIDR1](#): ID Register 1

[TRCIDR10](#): ID Register 10

[TRCIDR11](#): ID Register 11

[TRCIDR12](#): ID Register 12

[TRCIDR13](#): ID Register 13

[TRCIDR2](#): ID Register 2

[TRCIDR3](#): ID Register 3

[TRCIDR4](#): ID Register 4

[TRCIDR5](#): ID Register 5

[TRCIDR6](#): ID Register 6

[TRCIDR7](#): ID Register 7

[TRCIDR8](#): ID Register 8

[TRCIDR9](#): ID Register 9

[TRCIMSPEC0](#): IMP DEF Register 0

[TRCIMSPEC<n>](#): IMP DEF Register <n>

[TRCITCTRL](#): Integration Mode Control Register

[TRCLAR](#): Lock Access Register

[TRCLSR](#): Lock Status Register

[TRCOSLSR](#): Trace OS Lock Status Register

[TRCPDCR](#): PowerDown Control Register

[TRCPDSR](#): PowerDown Status Register

[TRCPIDR0](#): Peripheral Identification Register 0

[TRCPIDR1](#): Peripheral Identification Register 1

[TRCPIDR2](#): Peripheral Identification Register 2

[TRCPIDR3](#): Peripheral Identification Register 3

[TRCPIDR4](#): Peripheral Identification Register 4

[TRCPIDR5](#): Peripheral Identification Register 5

[TRCPIDR6](#): Peripheral Identification Register 6

[TRCPIDR7](#): Peripheral Identification Register 7

[TRCPRGCTRL](#): Programming Control Register

[TRCQCTRL](#): Q Element Control Register

[TRCRSCTRL<n>](#): Resource Selection Control Register <n>

[TRCRSR](#): Resources Status Register

[TRCSEQEVR<n>](#): Sequencer State Transition Control Register <n>
[TRCSEQRSTEV](#): Sequencer Reset Control Register
[TRCSEQSTR](#): Sequencer State Register
[TRCSSCCR<n>](#): Single-shot Comparator Control Register <n>
[TRCSSCSR<n>](#): Single-shot Comparator Control Status Register <n>
[TRCSSPCICR<n>](#): Single-shot Processing Element Comparator Input Control Register <n>
[TRCSTALLCTRL](#): Stall Control Register
[TRCSTATR](#): Trace Status Register
[TRCSYNCP](#): Synchronization Period Register
[TRCTRACEIDR](#): Trace ID Register
[TRCTSCTRL](#): Timestamp Control Register
[TRCVICTLR](#): ViewInst Main Control Register
[TRCVIICTRL](#): ViewInst Include/Exclude Control Register
[TRCVIPCSSLR](#): ViewInst Start/Stop PE Comparator Control Register
[TRCVISSCTRL](#): ViewInst Start/Stop Control Register
[TRCVMIDCCTRL0](#): Virtual Context Identifier Comparator Control Register 0
[TRCVMIDCCTRL1](#): Virtual Context Identifier Comparator Control Register 1
[TRCVMIDCVR<n>](#): Virtual Context Identifier Comparator Value Register <n>

30/09/2021 14:57

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

External register index by offset

Below are indexes for external registers in the following blocks:

- [GIC Distributor](#)
- [Debug](#)
- [GIC Virtual interface control](#)
- [PMU](#)
- [GIC Redistributor](#)
- [GIC Virtual CPU interface](#)
- [CTI](#)
- [GIC ITS control](#)
- [GIC CPU interface](#)
- [Timer](#)
- [GIC ITS translation](#)
- [AMU](#)
- [ETE](#)
- [MPAM](#)
- [RAS](#)

In the GIC Distributor block:

| Frame | Offset | Name | Description |
|-----------|------------------|--|---|
| Dist_base | 0x0000 | GICD_CTLR | Distributor Control Register |
| Dist_base | 0x0004 | GICD_TYPER | Interrupt Controller Type Register |
| Dist_base | 0x0008 | GICD_IIDR | Distributor Implementer Identification Register |
| Dist_base | 0x000C | GICD_TYPER2 | Interrupt Controller Type Register 2 |
| Dist_base | 0x0010 | GICD_STATUSR | Error Reporting Status Register |
| Dist_base | 0x0010 | GICD_STATUSR | Error Reporting Status Register |
| Dist_base | 0x0040 | GICD_SETSPI_NSR | Set Non-secure SPI Pending Register |
| Dist_base | 0x0048 | GICD_CLRSPI_NSR | Clear Non-secure SPI Pending Register |
| Dist_base | 0x0050 | GICD_SETSPI_SR | Set Secure SPI Pending Register |
| Dist_base | 0x0058 | GICD_CLRSPI_SR | Clear Secure SPI Pending Register |
| Dist_base | 0x0080 + (4 * n) | GICD_IGROUPR<n> | Interrupt Group Registers |
| Dist_base | 0x0100 + (4 * n) | GICD_ISENABLER<n> | Interrupt Set-Enable Registers |
| Dist_base | 0x0180 + (4 * n) | GICD_ICENABLER<n> | Interrupt Clear-Enable Registers |
| Dist_base | 0x0200 + (4 * n) | GICD_ISPENDR<n> | Interrupt Set-Pending Registers |
| Dist_base | 0x0280 + (4 * n) | GICD_ICPENDR<n> | Interrupt Clear-Pending Registers |
| Dist_base | 0x0300 + (4 * n) | GICD_ISACTIVER<n> | Interrupt Set-Active Registers |
| Dist_base | 0x0380 + (4 * n) | GICD_ICACTIVER<n> | Interrupt Clear-Active Registers |
| Dist_base | 0x0400 + (4 * n) | GICD_IPRIORITYR<n> | Interrupt Priority Registers |
| Dist_base | 0x0800 + (4 * n) | GICD_ITARGETSR<n> | Interrupt Processor Targets Registers |
| Dist_base | 0x0C00 + (4 * n) | GICD_ICFGR<n> | Interrupt Configuration Registers |
| Dist_base | 0x0D00 + (4 * n) | GICD_IGRPMODR<n> | Interrupt Group Modifier Registers |
| Dist_base | 0x0E00 + (4 * n) | GICD_NSACR<n> | Non-secure Access Control Registers |
| Dist_base | 0x0F00 | GICD_SGIR | Software Generated Interrupt Register |
| Dist_base | 0x0F10 + (4 * n) | GICD_CPENDSGIR<n> | SGI Clear-Pending Registers |
| Dist_base | 0x0F20 + (4 * n) | GICD_SPENDSGIR<n> | SGI Set-Pending Registers |
| Dist_base | 0x0F80 + (4 * n) | GICD_INMIR<n> | Non-maskable Interrupt Registers, x = 0 to 31 |
| Dist_base | 0x1000 + (4 * n) | GICD_IGROUPR<n>E | Interrupt Group Registers (extended SPI range) |
| Dist_base | 0x1200 + (4 * n) | GICD_ISENABLER<n>E | Interrupt Set-Enable Registers |

| Frame | Offset | Name | Description |
|-----------|------------------|---|---|
| Dist_base | 0x1400 + (4 * n) | GICD_ICENABLER<n>E | Interrupt Clear-Enable Registers |
| Dist_base | 0x1600 + (4 * n) | GICD_ISPENDR<n>E | Interrupt Set-Pending Registers (extended SPI range) |
| Dist_base | 0x1800 + (4 * n) | GICD_ICPENDR<n>E | Interrupt Clear-Pending Registers (extended SPI range) |
| Dist_base | 0x1A00 + (4 * n) | GICD_ISACTIVER<n>E | Interrupt Set-Active Registers (extended SPI range) |
| Dist_base | 0x1C00 + (4 * n) | GICD_ICACTIVER<n>E | Interrupt Clear-Active Registers (extended SPI range) |
| Dist_base | 0x2000 + (4 * n) | GICD_IPRIORITYR<n>E | Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC. |
| Dist_base | 0x3000 + (4 * n) | GICD_ICFGR<n>E | Interrupt Configuration Registers (Extended SPI Range) |
| Dist_base | 0x3400 + (4 * n) | GICD_IGRPMODR<n>E | Interrupt Group Modifier Registers (extended SPI range) |
| Dist_base | 0x3600 + (4 * n) | GICD_NSACR<n>E | Non-secure Access Control Registers |
| Dist_base | 0x3B00 + (4 * n) | GICD_INMIR<n>E | Non-maskable Interrupt Registers for Extended SPIs, x = 0 to 31 |
| Dist_base | 0x6000 + (8 * n) | GICD_IROUTER<n> | Interrupt Routing Registers |
| Dist_base | 0x8000 + (8 * n) | GICD_IROUTER<n>E | Interrupt Routing Registers (Extended SPI Range) |
| MSI_base | 0x0004 | GICM_TYPER | Distributor MSI Type Register |
| MSI_base | 0x0040 | GICM_SETSPI_NSR | Set Non-secure SPI Pending Register |
| MSI_base | 0x0048 | GICM_CLRSPI_NSR | Clear Non-secure SPI Pending Register |
| MSI_base | 0x0050 | GICM_SETSPI_SR | Set Secure SPI Pending Register |
| MSI_base | 0x0058 | GICM_CLRSPI_SR | Clear Secure SPI Pending Register |
| MSI_base | 0x0FCC | GICM_IIDR | Distributor Implementer Identification Register |

In the Debug block:

| Offset | Name | Description |
|------------------|---|---|
| 0x020 | EDES | External Debug Event Status Register |
| 0x024 | EDECR | External Debug Execution Control Register |
| 0x030 | EDWAR[31:0] | External Debug Watchpoint Address Register |
| 0x034 | EDWAR[63:32] | External Debug Watchpoint Address Register |
| 0x080 | DBGDTRRX_EL0 | Debug Data Transfer Register, Receive |
| 0x084 | EDITR | External Debug Instruction Transfer Register |
| 0x088 | EDSCR | External Debug Status and Control Register |
| 0x08C | DBGDTRTX_EL0 | Debug Data Transfer Register, Transmit |
| 0x090 | EDRCR | External Debug Reserve Control Register |
| 0x094 | EDACR | External Debug Auxiliary Control Register |
| 0x098 | EDECCR | External Debug Exception Catch Control Register |
| 0x0A0 | EDPCSR[31:0] | External Debug Program Counter Sample Register |
| 0x0A4 | EDCIDS | External Debug Context ID Sample Register |
| 0x0A8 | EDVIDSR | External Debug Virtual Context Sample Register |
| 0x0AC | EDPCSR[63:32] | External Debug Program Counter Sample Register |
| 0x300 | OSLAR_EL1 | OS Lock Access Register |
| 0x310 | EDPRCR | External Debug Power/Reset Control Register |
| 0x314 | EDPRSR | External Debug Processor Status Register |
| 0x400 + (16 * n) | DBGBVR<n>_EL1[63:0] | Debug Breakpoint Value Registers |
| 0x408 + (16 * n) | DBGBCR<n>_EL1 | Debug Breakpoint Control Registers |

| Offset | Name | Description |
|------------------|---|---|
| 0x800 + (16 * n) | DBGWVR<n>_EL1[63:0] | Debug Watchpoint Value Registers |
| 0x808 + (16 * n) | DBGWCR<n>_EL1 | Debug Watchpoint Control Registers |
| 0xD00 | MIDR_EL1 | Main ID Register |
| 0xD20 | EDPFR[31:0] | External Debug Processor Feature Register |
| 0xD24 | EDPFR[63:32] | External Debug Processor Feature Register |
| 0xD28 | EDDFR[31:0] | External Debug Feature Register |
| 0xD2C | EDDFR[63:32] | External Debug Feature Register |
| 0xD60 | EDAA32PFR | External Debug Auxiliary Processor Feature Register |
| 0xF00 | EDITCTRL | External Debug Integration mode Control register |
| 0xFA0 | DBGCLAIMSET_EL1 | Debug CLAIM Tag Set register |
| 0xFA4 | DBGCLAIMCLR_EL1 | Debug CLAIM Tag Clear register |
| 0xFA8 | EDDEVAFF0 | External Debug Device Affinity register 0 |
| 0xFAC | EDDEVAFF1 | External Debug Device Affinity register 1 |
| 0xFB0 | EDLAR | External Debug Lock Access Register |
| 0xFB4 | EDLSR | External Debug Lock Status Register |
| 0xFB8 | DBGAUTHSTATUS_EL1 | Debug Authentication Status register |
| 0xFBC | EDDEVARCH | External Debug Device Architecture register |
| 0xFC0 | EDDEVID2 | External Debug Device ID register 2 |
| 0xFC4 | EDDEVID1 | External Debug Device ID register 1 |
| 0xFC8 | EDDEVID | External Debug Device ID register 0 |
| 0xFCC | EDDEVTYPE | External Debug Device Type register |
| 0xFD0 | EDPIDR4 | External Debug Peripheral Identification Register 4 |
| 0xFE0 | EDPIDR0 | External Debug Peripheral Identification Register 0 |
| 0xFE4 | EDPIDR1 | External Debug Peripheral Identification Register 1 |
| 0xFE8 | EDPIDR2 | External Debug Peripheral Identification Register 2 |
| 0xFEC | EDPIDR3 | External Debug Peripheral Identification Register 3 |
| 0xFF0 | EDCIDR0 | External Debug Component Identification Register 0 |
| 0xFF4 | EDCIDR1 | External Debug Component Identification Register 1 |
| 0xFF8 | EDCIDR2 | External Debug Component Identification Register 2 |
| 0xFFC | EDCIDR3 | External Debug Component Identification Register 3 |

In the GIC Virtual interface control block:

| Offset | Name | Description |
|------------------|-----------------------------------|---------------------------------------|
| 0x0000 | GICH_HCR | Hypervisor Control Register |
| 0x0004 | GICH_VTR | Virtual Type Register |
| 0x0008 | GICH_VMCR | Virtual Machine Control Register |
| 0x0010 | GICH_MISR | Maintenance Interrupt Status Register |
| 0x0020 | GICH_EISR | End Interrupt Status Register |
| 0x0030 | GICH_ELRSR | Empty List Register Status Register |
| 0x00F0 + (4 * n) | GICH_APR<n> | Active Priorities Registers |
| 0x0100 + (4 * n) | GICH_LR<n> | List Registers |

In the PMU block:

| Offset | Name | Description |
|-----------------|---------------------------------------|--|
| 0x000 + (8 * n) | PMEVCNTR<n>_EL0 | Performance Monitors Event Count Registers |
| 0x0F8 | PMCCNTR_EL0[31:0] | Performance Monitors Cycle Counter |
| 0x0FC | PMCCNTR_EL0[63:32] | Performance Monitors Cycle Counter |

| Offset | Name | Description |
|-----------------|--|---|
| 0x200 | PMPCSR[31:0] | Program Counter Sample Register |
| 0x204 | PMPCSR[63:32] | Program Counter Sample Register |
| 0x208 | PMCID1SR | CONTEXTIDR_EL1 Sample Register |
| 0x20C | PMVIDSR | VMID Sample Register |
| 0x220 | PMPCSR[31:0] | Program Counter Sample Register |
| 0x224 | PMPCSR[63:32] | Program Counter Sample Register |
| 0x228 | PMCID1SR | CONTEXTIDR_EL1 Sample Register |
| 0x22C | PMCID2SR | CONTEXTIDR_EL2 Sample Register |
| 0x400 + (4 * n) | PMEVTYPER<n>_EL0 | Performance Monitors Event Type Registers |
| 0x47C | PMCCFILTR_EL0 | Performance Monitors Cycle Counter Filter Register |
| 0xA00 + (4 * n) | PMEVFILTR<n> | Performance Monitors Event Type Select Register <n>; |
| 0xC00 | PMCNTENSET_EL0 | Performance Monitors Count Enable Set register |
| 0xC20 | PMCNTENCLR_EL0 | Performance Monitors Count Enable Clear register |
| 0xC40 | PMINTENSET_EL1 | Performance Monitors Interrupt Enable Set register |
| 0xC60 | PMINTENCLR_EL1 | Performance Monitors Interrupt Enable Clear register |
| 0xC80 | PMOVSCLR_EL0 | Performance Monitors Overflow Flag Status Clear register |
| 0xCA0 | PMSWINC_EL0 | Performance Monitors Software Increment register |
| 0xCC0 | PMOVSSET_EL0 | Performance Monitors Overflow Flag Status Set register |
| 0xE00 | PMCFGR | Performance Monitors Configuration Register |
| 0xE04 | PMCR_EL0 | Performance Monitors Control Register |
| 0xE20 | PMCEID0 | Performance Monitors Common Event Identification register 0 |
| 0xE24 | PMCEID1 | Performance Monitors Common Event Identification register 1 |
| 0xE28 | PMCEID2 | Performance Monitors Common Event Identification register 2 |
| 0xE2C | PMCEID3 | Performance Monitors Common Event Identification register 3 |
| 0xE40 | PMMIR | Performance Monitors Machine Identification Register |
| 0xF00 | PMTCTRL | Performance Monitors Integration mode Control register |
| 0xFA8 | PMDEVAFF0 | Performance Monitors Device Affinity register 0 |
| 0xFAC | PMDEVAFF1 | Performance Monitors Device Affinity register 1 |
| 0xFB0 | PMLAR | Performance Monitors Lock Access Register |
| 0xFB4 | PMLSR | Performance Monitors Lock Status Register |
| 0xFB8 | PMAUTHSTATUS | Performance Monitors Authentication Status register |
| 0xFBC | PMDEVARCH | Performance Monitors Device Architecture register |
| 0xFC8 | PMDEVID | Performance Monitors Device ID register |
| 0xFCC | PMDEVTYPE | Performance Monitors Device Type register |
| 0xFD0 | PMPIDR4 | Performance Monitors Peripheral Identification Register 4 |
| 0xFE0 | PMPIDR0 | Performance Monitors Peripheral Identification Register 0 |
| 0xFE4 | PMPIDR1 | Performance Monitors Peripheral Identification Register 1 |
| 0xFE8 | PMPIDR2 | Performance Monitors Peripheral Identification Register 2 |
| 0xFEC | PMPIDR3 | Performance Monitors Peripheral Identification Register 3 |
| 0xFF0 | PMCIDR0 | Performance Monitors Component Identification Register 0 |
| 0xFF4 | PMCIDR1 | Performance Monitors Component Identification Register 1 |
| 0xFF8 | PMCIDR2 | Performance Monitors Component Identification Register 2 |
| 0xFFC | PMCIDR3 | Performance Monitors Component Identification Register 3 |

In the GIC Redistributor block:

| Frame | Offset | Name | Description |
|-----------|------------------|---|---|
| RD_base | 0x0000 | GICR_CTLR | Redistributor Control Register |
| RD_base | 0x0004 | GICR_IIDR | Redistributor Implementer Identification Register |
| RD_base | 0x0008 | GICR_TYPER | Redistributor Type Register |
| RD_base | 0x0010 | GICR_STATUSR | Error Reporting Status Register |
| RD_base | 0x0010 | GICR_STATUSR | Error Reporting Status Register |
| RD_base | 0x0014 | GICR_WAKER | Redistributor Wake Register |
| RD_base | 0x0018 | GICR_MPAMIDR | Report maximum PARTID and PMG Register |
| RD_base | 0x001C | GICR_PARTIDR | Set PARTID and PMG Register |
| RD_base | 0x0040 | GICR_SETLPIR | Set LPI Pending Register |
| RD_base | 0x0048 | GICR_CLRLPIR | Clear LPI Pending Register |
| RD_base | 0x0070 | GICR_PROPBASER | Redistributor Properties Base Address Register |
| RD_base | 0x0078 | GICR_PENDBASER | Redistributor LPI Pending Table Base Address Register |
| RD_base | 0x00A0 | GICR_INVLPIR | Redistributor Invalidate LPI Register |
| RD_base | 0x00B0 | GICR_INVALIDLR | Redistributor Invalidate All Register |
| RD_base | 0x00C0 | GICR_SYNCR | Redistributor Synchronize Register |
| SGI_base | 0x0080 | GICR_IGROUPR0 | Interrupt Group Register 0 |
| SGI_base | 0x0080 + (4 * n) | GICR_IGROUPR<n>E | Interrupt Group Registers |
| SGI_base | 0x0100 | GICR_ISENBALER0 | Interrupt Set-Enable Register 0 |
| SGI_base | 0x0100 + (4 * n) | GICR_ISENBALER<n>E | Interrupt Set-Enable Registers |
| SGI_base | 0x0180 | GICR_ICENABLER0 | Interrupt Clear-Enable Register 0 |
| SGI_base | 0x0180 + (4 * n) | GICR_ICENABLER<n>E | Interrupt Clear-Enable Registers |
| SGI_base | 0x0200 | GICR_ISPENDR0 | Interrupt Set-Pending Register 0 |
| SGI_base | 0x0200 + (4 * n) | GICR_ISPENDR<n>E | Interrupt Set-Pending Registers |
| SGI_base | 0x0280 | GICR_ICPENDR0 | Interrupt Clear-Pending Register 0 |
| SGI_base | 0x0280 + (4 * n) | GICR_ICPENDR<n>E | Interrupt Clear-Pending Registers |
| SGI_base | 0x0300 | GICR_ISACTIVER0 | Interrupt Set-Active Register 0 |
| SGI_base | 0x0300 + (4 * n) | GICR_ISACTIVER<n>E | Interrupt Set-Active Registers |
| SGI_base | 0x0380 | GICR_ICACTIVER0 | Interrupt Clear-Active Register 0 |
| SGI_base | 0x0380 + (4 * n) | GICR_ICACTIVER<n>E | Interrupt Clear-Active Registers |
| SGI_base | 0x0400 + (4 * n) | GICR_IPRIORITYR<n>E | Interrupt Priority Registers (extended PPI range) |
| SGI_base | 0x0400 + (4 * n) | GICR_IPRIORITYR<n> | Interrupt Priority Registers |
| SGI_base | 0x0C00 | GICR_ICFGR0 | Interrupt Configuration Register 0 |
| SGI_base | 0x0C00 + (4 * n) | GICR_ICFGR<n>E | Interrupt configuration registers |
| SGI_base | 0x0C04 | GICR_ICFGR1 | Interrupt Configuration Register 1 |
| SGI_base | 0x0D00 | GICR_IGRPMODR0 | Interrupt Group Modifier Register 0 |
| SGI_base | 0x0D00 + (4 * n) | GICR_IGRPMODR<n>E | Interrupt Group Modifier Registers |
| SGI_base | 0x0E00 | GICR_NSACR | Non-secure Access Control Register |
| SGI_base | 0x0F80 | GICR_INMIR0 | Non-maskable Interrupt Register for PPIs. |
| SGI_base | 0x0F80 + (4 * n) | GICR_INMIR<n>E | Non-maskable Interrupt Registers for Extended PPIs, x = 1 to 2. |
| VLPI_base | 0x0070 | GICR_VPROPBASER | Virtual Redistributor Properties Base Address Register |
| VLPI_base | 0x0078 | GICR_VPENDBASER | Virtual Redistributor LPI Pending Table Base Address Register |

| Frame | Offset | Name | Description |
|-----------|--------|--------------------------------|--|
| VLPI_base | 0x0080 | GICR_VSGIR | Redistributor virtual SGI pending state request register |
| VLPI_base | 0x0088 | GICR_VSGIPENDR | Redistributor virtual SGI pending state register |

In the GIC Virtual CPU interface block:

| Offset | Name | Description |
|------------------|-----------------------------------|---|
| 0x0000 | GICV_CTLR | Virtual Machine Control Register |
| 0x0004 | GICV_PMR | Virtual Machine Priority Mask Register |
| 0x0008 | GICV_BPR | Virtual Machine Binary Point Register |
| 0x000C | GICV_IAR | Virtual Machine Interrupt Acknowledge Register |
| 0x0010 | GICV_EOIR | Virtual Machine End Of Interrupt Register |
| 0x0014 | GICV_RPR | Virtual Machine Running Priority Register |
| 0x0018 | GICV_HPPIR | Virtual Machine Highest Priority Pending Interrupt Register |
| 0x001C | GICV_ABPR | Virtual Machine Aliased Binary Point Register |
| 0x0020 | GICV_AIAR | Virtual Machine Aliased Interrupt Acknowledge Register |
| 0x0024 | GICV_AEOIR | Virtual Machine Aliased End Of Interrupt Register |
| 0x0028 | GICV_AHPPIR | Virtual Machine Aliased Highest Priority Pending Interrupt Register |
| 0x002C | GICV_STATUSR | Virtual Machine Error Reporting Status Register |
| 0x00D0 + (4 * n) | GICV_APR<n> | Virtual Machine Active Priorities Registers |
| 0x00FC | GICV_IIDR | Virtual Machine CPU Interface Identification Register |
| 0x1000 | GICV_DIR | Virtual Machine Deactivate Interrupt Register |

In the CTI block:

| Offset | Name | Description |
|-----------------|-----------------------------------|--|
| 0x000 | CTICONTROL | CTI Control register |
| 0x010 | CTIINTACK | CTI Output Trigger Acknowledge register |
| 0x014 | CTIAPPSET | CTI Application Trigger Set register |
| 0x018 | CTIAPPCLEAR | CTI Application Trigger Clear register |
| 0x01C | CTIAPPPULSE | CTI Application Pulse register |
| 0x020 + (4 * n) | CTIINEN<n> | CTI Input Trigger to Output Channel Enable registers |
| 0x0A0 + (4 * n) | CTIOUTEN<n> | CTI Input Channel to Output Trigger Enable registers |
| 0x130 | CTITRIGINSTATUS | CTI Trigger In Status register |
| 0x134 | CTITRIGOUTSTATUS | CTI Trigger Out Status register |
| 0x138 | CTICHINSTATUS | CTI Channel In Status register |
| 0x13C | CTICHOUTSTATUS | CTI Channel Out Status register |
| 0x140 | CTIGATE | CTI Channel Gate Enable register |
| 0x144 | ASICCTL | CTI External Multiplexer Control register |
| 0x150 | CTIDEVCTL | CTI Device Control register |
| 0xF00 | CTIITCTRL | CTI Integration mode Control register |
| 0xFA0 | CTICLAIMSET | CTI CLAIM Tag Set register |
| 0xFA4 | CTICLAIMCLR | CTI CLAIM Tag Clear register |
| 0xFA8 | CTIDEVAFF0 | CTI Device Affinity register 0 |
| 0xFAC | CTIDEVAFF1 | CTI Device Affinity register 1 |
| 0xFB0 | CTILAR | CTI Lock Access Register |
| 0xFB4 | CTILSR | CTI Lock Status Register |
| 0xFB8 | CTIAUTHSTATUS | CTI Authentication Status register |

| Offset | Name | Description |
|--------|----------------------------|--|
| 0xFBC | CTIDEVARCH | CTI Device Architecture register |
| 0xFC0 | CTIDEVID2 | CTI Device ID register 2 |
| 0xFC4 | CTIDEVID1 | CTI Device ID register 1 |
| 0xFC8 | CTIDEVID | CTI Device ID register 0 |
| 0xFCC | CTIDEVTYPE | CTI Device Type register |
| 0xFD0 | CTIPIDR4 | CTI Peripheral Identification Register 4 |
| 0xFE0 | CTIPIDR0 | CTI Peripheral Identification Register 0 |
| 0xFE4 | CTIPIDR1 | CTI Peripheral Identification Register 1 |
| 0xFE8 | CTIPIDR2 | CTI Peripheral Identification Register 2 |
| 0xFEC | CTIPIDR3 | CTI Peripheral Identification Register 3 |
| 0xFF0 | CTICIDR0 | CTI Component Identification Register 0 |
| 0xFF4 | CTICIDR1 | CTI Component Identification Register 1 |
| 0xFF8 | CTICIDR2 | CTI Component Identification Register 2 |
| 0xFFC | CTICIDR3 | CTI Component Identification Register 3 |

In the GIC ITS control block:

| Offset | Name | Description |
|------------------|-------------------------------------|--|
| 0x0000 | GITS_CTLR | ITS Control Register |
| 0x0004 | GITS_IIDR | ITS Identification Register |
| 0x0008 | GITS_TYPER | ITS Type Register |
| 0x0010 | GITS_MPAMIDR | Report maximum PARTID and PMG Register |
| 0x0014 | GITS_PARTIDR | Set PARTID and PMG Register |
| 0x0018 | GITS_MPIDR | Report ITS's affinity. |
| 0x0040 | GITS_STATUSR | ITS Error Reporting Status Register |
| 0x0048 | GITS_UMSIR | ITS Unmapped MSI register |
| 0x0080 | GITS_CBASER | ITS Command Queue Descriptor |
| 0x0088 | GITS_CWRITER | ITS Write Register |
| 0x0090 | GITS_CREADR | ITS Read Register |
| 0x0100 + (8 * n) | GITS_BASER<n> | ITS Translation Table Descriptors |
| 0x20020 | GITS_SGIR | ITS SGI Register |

In the GIC CPU interface block:

| Offset | Name | Description |
|------------------|-----------------------------------|---|
| 0x0000 | GICC_CTLR | CPU Interface Control Register |
| 0x0004 | GICC_PMR | CPU Interface Priority Mask Register |
| 0x0008 | GICC_BPR | CPU Interface Binary Point Register |
| 0x000C | GICC_IAR | CPU Interface Interrupt Acknowledge Register |
| 0x0010 | GICC_EOIR | CPU Interface End Of Interrupt Register |
| 0x0014 | GICC_RPR | CPU Interface Running Priority Register |
| 0x0018 | GICC_HPPIR | CPU Interface Highest Priority Pending Interrupt Register |
| 0x001C | GICC_ABPR | CPU Interface Aliased Binary Point Register |
| 0x0020 | GICC_AIAR | CPU Interface Aliased Interrupt Acknowledge Register |
| 0x0024 | GICC_AEOIR | CPU Interface Aliased End Of Interrupt Register |
| 0x0028 | GICC_AHPPIR | CPU Interface Aliased Highest Priority Pending Interrupt Register |
| 0x002C | GICC_STATUSR | CPU Interface Status Register |
| 0x002C | GICC_STATUSR | CPU Interface Status Register |
| 0x00D0 + (4 * n) | GICC_APR<n> | CPU Interface Active Priorities Registers |

| Offset | Name | Description |
|--------------------|-------------------------------------|--|
| $0x00E0 + (4 * n)$ | GICC_NSAPR<n> | CPU Interface Non-secure Active Priorities Registers |
| $0x00FC$ | GICC_IIDR | CPU Interface Identification Register |
| $0x1000$ | GICC_DIR | CPU Interface Deactivate Interrupt Register |

In the Timer block:

| Frame | Offset | Name | Description |
|----------------|-------------------|---|---|
| CNTBaseN | $0x000$ | CNTPCT[31:0] | Counter-timer Physical Count |
| CNTBaseN | $0x004$ | CNTPCT[63:32] | Counter-timer Physical Count |
| CNTBaseN | $0x008$ | CNTVCT[31:0] | Counter-timer Virtual Count |
| CNTBaseN | $0x00C$ | CNTVCT[63:32] | Counter-timer Virtual Count |
| CNTBaseN | $0x010$ | CNTFRQ | Counter-timer Frequency |
| CNTBaseN | $0x014$ | CNTEL0ACR | Counter-timer EL0 Access Control Register |
| CNTBaseN | $0x018$ | CNTVOFF[31:0] | Counter-timer Virtual Offset |
| CNTBaseN | $0x01C$ | CNTVOFF[63:32] | Counter-timer Virtual Offset |
| CNTBaseN | $0x020$ | CNTP_CVAL[31:0] | Counter-timer Physical Timer CompareValue |
| CNTBaseN | $0x024$ | CNTP_CVAL[63:32] | Counter-timer Physical Timer CompareValue |
| CNTBaseN | $0x028$ | CNTP_TVAL | Counter-timer Physical Timer TimerValue |
| CNTBaseN | $0x02C$ | CNTP_CTL | Counter-timer Physical Timer Control |
| CNTBaseN | $0x030$ | CNTV_CVAL[31:0] | Counter-timer Virtual Timer CompareValue |
| CNTBaseN | $0x034$ | CNTV_CVAL[63:32] | Counter-timer Virtual Timer CompareValue |
| CNTBaseN | $0x038$ | CNTV_TVAL | Counter-timer Virtual Timer TimerValue |
| CNTBaseN | $0x03C$ | CNTV_CTL | Counter-timer Virtual Timer Control |
| CNTBaseN | $0xFD0 + (4 * n)$ | CounterID<n> | Counter ID registers |
| CNTCTLBase | $0x000$ | CNTFRQ | Counter-timer Frequency |
| CNTCTLBase | $0x004$ | CNTNSAR | Counter-timer Non-secure Access Register |
| CNTCTLBase | $0x008$ | CNTTIDR | Counter-timer Timer ID Register |
| CNTCTLBase | $0x040 + (4 * n)$ | CNTACR<n> | Counter-timer Access Control Registers |
| CNTCTLBase | $0x080 + (8 * n)$ | CNTVOFF<n>[31:0] | Counter-timer Virtual Offsets |
| CNTCTLBase | $0x084 + (8 * n)$ | CNTVOFF<n>[63:32] | Counter-timer Virtual Offsets |
| CNTCTLBase | $0xFD0 + (4 * n)$ | CounterID<n> | Counter ID registers |
| CNTControlBase | $0x000$ | CNTCR | Counter Control Register |
| CNTControlBase | $0x004$ | CNTSR | Counter Status Register |
| CNTControlBase | $0x008$ | CNTCV[63:0] | Counter Count Value register |
| CNTControlBase | $0x020$ | CNTFID0 | Counter Frequency ID |
| CNTControlBase | $0x020 + (4 * n)$ | CNTFID<n> | Counter Frequency IDs, $n > 0$ |
| CNTControlBase | $0x10$ | CNTSCR | Counter Scale Register |
| CNTControlBase | $0x1C$ | CNTID | Counter Identification Register |
| CNTControlBase | $0xFD0 + (4 * n)$ | CounterID<n> | Counter ID registers |
| CNTEL0BaseN | $0x000$ | CNTPCT[31:0] | Counter-timer Physical Count |
| CNTEL0BaseN | $0x004$ | CNTPCT[63:32] | Counter-timer Physical Count |
| CNTEL0BaseN | $0x008$ | CNTVCT[31:0] | Counter-timer Virtual Count |
| CNTEL0BaseN | $0x00C$ | CNTVCT[63:32] | Counter-timer Virtual Count |
| CNTEL0BaseN | $0x010$ | CNTFRQ | Counter-timer Frequency |

| Frame | Offset | Name | Description |
|-------------|-----------------|------------------------------------|---|
| CNTELOBaseN | 0x020 | CNTP_CVAL[31:0] | Counter-timer Physical Timer CompareValue |
| CNTELOBaseN | 0x024 | CNTP_CVAL[63:32] | Counter-timer Physical Timer CompareValue |
| CNTELOBaseN | 0x028 | CNTP_TVAL | Counter-timer Physical Timer TimerValue |
| CNTELOBaseN | 0x02C | CNTP_CTL | Counter-timer Physical Timer Control |
| CNTELOBaseN | 0x030 | CNTV_CVAL[31:0] | Counter-timer Virtual Timer CompareValue |
| CNTELOBaseN | 0x034 | CNTV_CVAL[63:32] | Counter-timer Virtual Timer CompareValue |
| CNTELOBaseN | 0x038 | CNTV_TVAL | Counter-timer Virtual Timer TimerValue |
| CNTELOBaseN | 0x03C | CNTV_CTL | Counter-timer Virtual Timer Control |
| CNTELOBaseN | 0xFD0 + (4 * n) | CounterID<n> | Counter ID registers |
| CNTReadBase | 0x000 | CNTCV[63:0] | Counter Count Value register |
| CNTReadBase | 0xFD0 + (4 * n) | CounterID<n> | Counter ID registers |

In the GIC ITS translation block:

| Offset | Name | Description |
|--------|---------------------------------|--------------------------|
| 0x0040 | GITS_TRANSLATER | ITS Translation Register |

In the AMU block:

| Offset | Name | Description |
|-----------------|---|--|
| 0x000 + (8 * n) | AMEVCNTR0<n>[31:0] | Activity Monitors Event Counter Registers 0 |
| 0x004 + (8 * n) | AMEVCNTR0<n>[63:32] | Activity Monitors Event Counter Registers 0 |
| 0x100 + (8 * n) | AMEVCNTR1<n>[31:0] | Activity Monitors Event Counter Registers 1 |
| 0x104 + (8 * n) | AMEVCNTR1<n>[63:32] | Activity Monitors Event Counter Registers 1 |
| 0x400 + (4 * n) | AMEVTYPER0<n> | Activity Monitors Event Type Registers 0 |
| 0x480 + (4 * n) | AMEVTYPER1<n> | Activity Monitors Event Type Registers 1 |
| 0xC00 | AMCNTENSET0 | Activity Monitors Count Enable Set Register 0 |
| 0xC04 | AMCNTENSET1 | Activity Monitors Count Enable Set Register 1 |
| 0xC20 | AMCNTENCLR0 | Activity Monitors Count Enable Clear Register 0 |
| 0xC24 | AMCNTENCLR1 | Activity Monitors Count Enable Clear Register 1 |
| 0xCE0 | AMCGCR | Activity Monitors Counter Group Configuration Register |
| 0xE00 | AMCFGR | Activity Monitors Configuration Register |
| 0xE04 | AMCR | Activity Monitors Control Register |
| 0xE08 | AMIIDR | Activity Monitors Implementation Identification Register |
| 0xFA8 | AMDEVAFF0 | Activity Monitors Device Affinity Register 0 |
| 0xFAC | AMDEVAFF1 | Activity Monitors Device Affinity Register 1 |
| 0xFBC | AMDEVARCH | Activity Monitors Device Architecture Register |
| 0xFCC | AMDEVTYPE | Activity Monitors Device Type Register |
| 0xFD0 | AMPIDR4 | Activity Monitors Peripheral Identification Register 4 |
| 0xFE0 | AMPIDR0 | Activity Monitors Peripheral Identification Register 0 |
| 0xFE4 | AMPIDR1 | Activity Monitors Peripheral Identification Register 1 |
| 0xFE8 | AMPIDR2 | Activity Monitors Peripheral Identification Register 2 |
| 0xFEC | AMPIDR3 | Activity Monitors Peripheral Identification Register 3 |
| 0xFF0 | AMCIDR0 | Activity Monitors Component Identification Register 0 |
| 0xFF4 | AMCIDR1 | Activity Monitors Component Identification Register 1 |
| 0xFF8 | AMCIDR2 | Activity Monitors Component Identification Register 2 |

| Offset | Name | Description |
|--------|-------------------------|---|
| 0xFFC | AMCIDR3 | Activity Monitors Component Identification Register 3 |

In the ETE block:

| Offset | Name | Description |
|-----------------|--------------------------------------|--|
| 0x004 | TRCPRGCTLR | Programming Control Register |
| 0x00C | TRCSTATR | Trace Status Register |
| 0x010 | TRCCONFIGR | Trace Configuration Register |
| 0x018 | TRCAUXCTLR | Auxiliary Control Register |
| 0x020 | TRCEVENTCTL0R | Event Control 0 Register |
| 0x024 | TRCEVENTCTL1R | Event Control 1 Register |
| 0x028 | TRCRSR | Resources Status Register |
| 0x02C | TRCSTALLCTLR | Stall Control Register |
| 0x030 | TRCTSCTLR | Timestamp Control Register |
| 0x034 | TRCSYNCPR | Synchronization Period Register |
| 0x038 | TRCCCCTLR | Cycle Count Control Register |
| 0x03C | TRCBBCTLR | Branch Broadcast Control Register |
| 0x040 | TRCTRACEIDR | Trace ID Register |
| 0x044 | TRCQCTLR | Q Element Control Register |
| 0x080 | TRCVICTLR | ViewInst Main Control Register |
| 0x084 | TRCVIIECTLR | ViewInst Include/Exclude Control Register |
| 0x088 | TRCVISSCTLR | ViewInst Start/Stop Control Register |
| 0x08C | TRCVIPCSSCTLR | ViewInst Start/Stop PE Comparator Control Register |
| 0x100 + (4 * n) | TRCSEQEVR<n> | Sequencer State Transition Control Register <n>; |
| 0x118 | TRCSEQRSTEV | Sequencer Reset Control Register |
| 0x11C | TRCSEQSTR | Sequencer State Register |
| 0x120 + (4 * n) | TRCEXTINSEL<n> | External Input Select Register <n>; |
| 0x140 + (4 * n) | TRCCNTRLDVR<n> | Counter Reload Value Register <n>; |
| 0x150 + (4 * n) | TRCCNTCTLR<n> | Counter Control Register <n>; |
| 0x160 + (4 * n) | TRCCNTVR<n> | Counter Value Register <n>; |
| 0x180 | TRCIDR8 | ID Register 8 |
| 0x184 | TRCIDR9 | ID Register 9 |
| 0x188 | TRCIDR10 | ID Register 10 |
| 0x18C | TRCIDR11 | ID Register 11 |
| 0x190 | TRCIDR12 | ID Register 12 |
| 0x194 | TRCIDR13 | ID Register 13 |
| 0x1C0 | TRCIMSPEC0 | IMP DEF Register 0 |
| 0x1C0 + (4 * n) | TRCIMSPEC<n> | IMP DEF Register <n>; |
| 0x1E0 | TRCIDR0 | ID Register 0 |
| 0x1E4 | TRCIDR1 | ID Register 1 |
| 0x1E8 | TRCIDR2 | ID Register 2 |
| 0x1EC | TRCIDR3 | ID Register 3 |
| 0x1F0 | TRCIDR4 | ID Register 4 |
| 0x1F4 | TRCIDR5 | ID Register 5 |
| 0x1F8 | TRCIDR6 | ID Register 6 |
| 0x1FC | TRCIDR7 | ID Register 7 |
| 0x200 + (4 * n) | TRCRSCTLR<n> | Resource Selection Control Register <n>; |
| 0x280 + (4 * n) | TRCSSCCR<n> | Single-shot Comparator Control Register <n>; |
| 0x2A0 + (4 * n) | TRCSSCSR<n> | Single-shot Comparator Control Status Register <n>; |

| Offset | Name | Description |
|-----------------|-------------------------------------|---|
| 0x2C0 + (4 * n) | TRCSSPCICR<n> | Single-shot Processing Element Comparator Input Control Register <n>; |
| 0x304 | TRCOSLSR | Trace OS Lock Status Register |
| 0x310 | TRCPDCR | PowerDown Control Register |
| 0x314 | TRCPDSR | PowerDown Status Register |
| 0x400 + (8 * n) | TRCACVR<n> | Address Comparator Value Register <n>; |
| 0x480 + (8 * n) | TRCACATR<n> | Address Comparator Access Type Register <n>; |
| 0x600 + (8 * n) | TRCCIDCVR<n> | Context Identifier Comparator Value Registers <n>; |
| 0x640 + (8 * n) | TRCVMIDCVR<n> | Virtual Context Identifier Comparator Value Register <n>; |
| 0x680 | TRCCIDCCTLR0 | Context Identifier Comparator Control Register 0 |
| 0x684 | TRCCIDCCTLR1 | Context Identifier Comparator Control Register 1 |
| 0x688 | TRCVMIDCCTLR0 | Virtual Context Identifier Comparator Control Register 0 |
| 0x68C | TRCVMIDCCTLR1 | Virtual Context Identifier Comparator Control Register 1 |
| 0xF00 | TRCITCTRL | Integration Mode Control Register |
| 0xFA0 | TRCCLAIMSET | Claim Tag Set Register |
| 0xFA4 | TRCCLAIMCLR | Claim Tag Clear Register |
| 0xFA8 | TRCDEVAFF | Device Affinity Register |
| 0xFB0 | TRCLAR | Lock Access Register |
| 0xFB4 | TRCLSR | Lock Status Register |
| 0xFB8 | TRCAUTHSTATUS | Authentication Status Register |
| 0xFBC | TRCDEVARCH | Device Architecture Register |
| 0xFC0 | TRCDEVID2 | Device Configuration Register 2 |
| 0xFC4 | TRCDEVID1 | Device Configuration Register 1 |
| 0xFC8 | TRCDEVID | Device Configuration Register |
| 0xFCC | TRCDEVTYPE | Device Type Register |
| 0xFD0 | TRCPIDR4 | Peripheral Identification Register 4 |
| 0xFD4 | TRCPIDR5 | Peripheral Identification Register 5 |
| 0xFD8 | TRCPIDR6 | Peripheral Identification Register 6 |
| 0xFDC | TRCPIDR7 | Peripheral Identification Register 7 |
| 0xFE0 | TRCPIDR0 | Peripheral Identification Register 0 |
| 0xFE4 | TRCPIDR1 | Peripheral Identification Register 1 |
| 0xFE8 | TRCPIDR2 | Peripheral Identification Register 2 |
| 0xFEC | TRCPIDR3 | Peripheral Identification Register 3 |
| 0xFF0 | TRCCIDR0 | Component Identification Register 0 |
| 0xFF4 | TRCCIDR1 | Component Identification Register 1 |
| 0xFF8 | TRCCIDR2 | Component Identification Register 2 |
| 0xFFC | TRCCIDR3 | Component Identification Register 3 |

In the MPAM block:

| Frame | Offset | Name | Description |
|---------------|--------|--------------------------------|---|
| MPAMF_BASE_ns | 0x0000 | MPAMF_IDR | MPAM Features Identification Register |
| MPAMF_BASE_ns | 0x0018 | MPAMF_IIDR | MPAM Implementation Identification Register |
| MPAMF_BASE_ns | 0x0020 | MPAMF_AIDR | MPAM Architecture Identification Register |
| MPAMF_BASE_ns | 0x0028 | MPAMF_IMPL_IDR | MPAM Implementation-Specific Partitioning Feature Identification Register |

| Frame | Offset | Name | Description |
|---------------|--------|--------------------------------------|--|
| MPAMF_BASE_ns | 0x0030 | MPAMF_CPOR_IDR | MPAM Features Cache Portion Partitioning ID register |
| MPAMF_BASE_ns | 0x0038 | MPAMF_CCAP_IDR | MPAM Features Cache Capacity Partitioning ID register |
| MPAMF_BASE_ns | 0x0040 | MPAMF_MBW_IDR | MPAM Memory Bandwidth Partitioning Identification Register |
| MPAMF_BASE_ns | 0x0048 | MPAMF_PRI_IDR | MPAM Priority Partitioning Identification Register |
| MPAMF_BASE_ns | 0x0050 | MPAMF_PARTID_NRW_IDR | MPAM PARTID Narrowing ID register |
| MPAMF_BASE_ns | 0x0080 | MPAMF_MSMON_IDR | MPAM Resource Monitoring Identification Register |
| MPAMF_BASE_ns | 0x0088 | MPAMF_CSUMON_IDR | MPAM Features Cache Storage Usage Monitoring ID register |
| MPAMF_BASE_ns | 0x0090 | MPAMF_MBWUMON_IDR | MPAM Features Memory Bandwidth Usage Monitoring ID register |
| MPAMF_BASE_ns | 0x00DC | MPAMF_ERR_MSI_MPAM | MPAM Error MSI Write MPAM Information Register |
| MPAMF_BASE_ns | 0x00E0 | MPAMF_ERR_MSI_ADDR_L | MPAM Error MSI Low-part Address Register |
| MPAMF_BASE_ns | 0x00E4 | MPAMF_ERR_MSI_ADDR_H | MPAM Error MSI High-part Address Register |
| MPAMF_BASE_ns | 0x00E8 | MPAMF_ERR_MSI_DATA | MPAM Error MSI Data Register |
| MPAMF_BASE_ns | 0x00EC | MPAMF_ERR_MSI_ATTR | MPAM Error MSI Write Attributes Register |
| MPAMF_BASE_ns | 0x00F0 | MPAMF_ECR | MPAM Error Control Register |
| MPAMF_BASE_ns | 0x00F8 | MPAMF_ESR | MPAM Error Status Register |
| MPAMF_BASE_ns | 0x0100 | MPAMCFG_PART_SEL | MPAM Partition Configuration Selection Register |
| MPAMF_BASE_ns | 0x0108 | MPAMCFG_CMAX | MPAM Cache Maximum Capacity Partition Configuration Register |
| MPAMF_BASE_ns | 0x0110 | MPAMCFG_CMIN | MPAM Cache Minimum Capacity Partition Configuration Register |
| MPAMF_BASE_ns | 0x0118 | MPAMCFG_CASSOC | MPAM Cache Maximum Associativity Partition Configuration Register |
| MPAMF_BASE_ns | 0x0200 | MPAMCFG_MBW_MIN | MPAM Memory Bandwidth Minimum Partition Configuration Register |
| MPAMF_BASE_ns | 0x0208 | MPAMCFG_MBW_MAX | MPAM Memory Bandwidth Maximum Partition Configuration Register |
| MPAMF_BASE_ns | 0x0220 | MPAMCFG_MBW_WINWD | MPAM Memory Bandwidth Partitioning Window Width Configuration Register |
| MPAMF_BASE_ns | 0x0300 | MPAMCFG_EN | MPAM Partition Configuration Enable Register |

| Frame | Offset | Name | Description |
|---------------|--------|--|--|
| MPAMF_BASE_ns | 0x0310 | MPAMCFG_DIS | MPAM Partition Configuration Disable Register |
| MPAMF_BASE_ns | 0x0320 | MPAMCFG_EN_FLAGS | MPAM Partition Configuration Enable Flags Register |
| MPAMF_BASE_ns | 0x0400 | MPAMCFG_PRI | MPAM Priority Partition Configuration Register |
| MPAMF_BASE_ns | 0x0500 | MPAMCFG_MBW_PROP | MPAM Memory Bandwidth Proportional Stride Partition Configuration Register |
| MPAMF_BASE_ns | 0x0600 | MPAMCFG_INTPARTID | MPAM Internal PARTID Narrowing Configuration Register |
| MPAMF_BASE_ns | 0x0800 | MSMON_CFG_MON_SEL | MPAM Monitor Instance Selection Register |
| MPAMF_BASE_ns | 0x0808 | MSMON_CAPT_EVNT | MPAM Capture Event Generation Register |
| MPAMF_BASE_ns | 0x0810 | MSMON_CFG_CSU_FLT | MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register |
| MPAMF_BASE_ns | 0x0818 | MSMON_CFG_CSU_CTL | MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register |
| MPAMF_BASE_ns | 0x0820 | MSMON_CFG_MBWU_FLT | MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register |
| MPAMF_BASE_ns | 0x0828 | MSMON_CFG_MBWU_CTL | MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register |
| MPAMF_BASE_ns | 0x0840 | MSMON_CSU | MPAM Cache Storage Usage Monitor Register |
| MPAMF_BASE_ns | 0x0848 | MSMON_CSU_CAPTURE | MPAM Cache Storage Usage Monitor Capture Register |
| MPAMF_BASE_ns | 0x0858 | MSMON_CSU_OFSR | MPAM CSU Monitor Overflow Status Register |
| MPAMF_BASE_ns | 0x0860 | MSMON_MBWU | MPAM Memory Bandwidth Usage Monitor Register |
| MPAMF_BASE_ns | 0x0868 | MSMON_MBWU_CAPTURE | MPAM Memory Bandwidth Usage Monitor Capture Register |
| MPAMF_BASE_ns | 0x0880 | MSMON_MBWU_L | MPAM Long Memory Bandwidth Usage Monitor Register |
| MPAMF_BASE_ns | 0x0890 | MSMON_MBWU_L_CAPTURE | MPAM Long Memory Bandwidth Usage Monitor Capture Register |
| MPAMF_BASE_ns | 0x0898 | MSMON_MBWU_OFSR | MPAM MBWU Monitor Overflow Status Register |
| MPAMF_BASE_ns | 0x08DC | MSMON_OFLOW_MSI_MPAM | MPAM Monitor Overflow MSI Write MPAM Information Register |
| MPAMF_BASE_ns | 0x08E0 | MSMON_OFLOW_MSI_ADDR_L | MPAM Monitor Overflow MSI Low-part Address Register |

| Frame | Offset | Name | Description |
|---------------|------------------|--|---|
| MPAMF_BASE_ns | 0x08E4 | MSMON_OFLOW_MSI_ADDR_H | MPAM Monitor Overflow MSI Write High-part Address Register |
| MPAMF_BASE_ns | 0x08E8 | MSMON_OFLOW_MSI_DATA | MPAM Monitor Overflow MSI Write Data Register |
| MPAMF_BASE_ns | 0x08EC | MSMON_OFLOW_MSI_ATTR | MPAM Monitor Overflow MSI Write Attributes Register |
| MPAMF_BASE_ns | 0x08F0 | MSMON_OFLOW_SR | MPAM Monitor Overflow Status Register |
| MPAMF_BASE_ns | 0x1000 + (4 * n) | MPAMCFG_CPBM<n> | MPAM Cache Portion Bitmap Partition Configuration Register |
| MPAMF_BASE_ns | 0x2000 + (4 * n) | MPAMCFG_MBW_PBM<n> | MPAM Bandwidth Portion Bitmap Partition Configuration Register |
| MPAMF_BASE_ri | 0x0000 | MPAMF_IDR | MPAM Features Identification Register |
| MPAMF_BASE_ri | 0x0018 | MPAMF_IIDR | MPAM Implementation Identification Register |
| MPAMF_BASE_ri | 0x0020 | MPAMF_AIDR | MPAM Architecture Identification Register |
| MPAMF_BASE_ri | 0x0028 | MPAMF_IMPL_IDR | MPAM Implementation-Specific Partitioning Feature Identification Register |
| MPAMF_BASE_ri | 0x0030 | MPAMF_CPOR_IDR | MPAM Features Cache Portion Partitioning ID register |
| MPAMF_BASE_ri | 0x0038 | MPAMF_CCAP_IDR | MPAM Features Cache Capacity Partitioning ID register |
| MPAMF_BASE_ri | 0x0040 | MPAMF_MBW_IDR | MPAM Memory Bandwidth Partitioning Identification Register |
| MPAMF_BASE_ri | 0x0048 | MPAMF_PRI_IDR | MPAM Priority Partitioning Identification Register |
| MPAMF_BASE_ri | 0x0050 | MPAMF_PARTID_NRW_IDR | MPAM PARTID Narrowing ID register |
| MPAMF_BASE_ri | 0x0080 | MPAMF_MSMON_IDR | MPAM Resource Monitoring Identification Register |
| MPAMF_BASE_ri | 0x0088 | MPAMF_CSUMON_IDR | MPAM Features Cache Storage Usage Monitoring ID register |
| MPAMF_BASE_ri | 0x0090 | MPAMF_MBWUMON_IDR | MPAM Features Memory Bandwidth Usage Monitoring ID register |
| MPAMF_BASE_ri | 0x00DC | MPAMF_ERR_MSI_MPAM | MPAM Error MSI Write MPAM Information Register |
| MPAMF_BASE_ri | 0x00E0 | MPAMF_ERR_MSI_ADDR_L | MPAM Error MSI Low-part Address Register |
| MPAMF_BASE_ri | 0x00E4 | MPAMF_ERR_MSI_ADDR_H | MPAM Error MSI High-part Address Register |
| MPAMF_BASE_ri | 0x00E8 | MPAMF_ERR_MSI_DATA | MPAM Error MSI Data Register |
| MPAMF_BASE_ri | 0x00EC | MPAMF_ERR_MSI_ATTR | MPAM Error MSI Write Attributes Register |
| MPAMF_BASE_ri | 0x00F0 | MPAMF_ECR | MPAM Error Control Register |

| Frame | Offset | Name | Description |
|---------------|--------|------------------------------------|--|
| MPAMF_BASE_rl | 0x00F8 | MPAMF_ESR | MPAM Error Status Register |
| MPAMF_BASE_rl | 0x0100 | MPAMCFG_PART_SEL | MPAM Partition Configuration Selection Register |
| MPAMF_BASE_rl | 0x0108 | MPAMCFG_CMAX | MPAM Cache Maximum Capacity Partition Configuration Register |
| MPAMF_BASE_rl | 0x0110 | MPAMCFG_CMIN | MPAM Cache Minimum Capacity Partition Configuration Register |
| MPAMF_BASE_rl | 0x0118 | MPAMCFG_CASSOC | MPAM Cache Maximum Associativity Partition Configuration Register |
| MPAMF_BASE_rl | 0x0200 | MPAMCFG_MBW_MIN | MPAM Memory Bandwidth Minimum Partition Configuration Register |
| MPAMF_BASE_rl | 0x0208 | MPAMCFG_MBW_MAX | MPAM Memory Bandwidth Maximum Partition Configuration Register |
| MPAMF_BASE_rl | 0x0220 | MPAMCFG_MBW_WINWD | MPAM Memory Bandwidth Partitioning Window Width Configuration Register |
| MPAMF_BASE_rl | 0x0300 | MPAMCFG_EN | MPAM Partition Configuration Enable Register |
| MPAMF_BASE_rl | 0x0310 | MPAMCFG_DIS | MPAM Partition Configuration Disable Register |
| MPAMF_BASE_rl | 0x0320 | MPAMCFG_EN_FLAGS | MPAM Partition Configuration Enable Flags Register |
| MPAMF_BASE_rl | 0x0400 | MPAMCFG_PRI | MPAM Priority Partition Configuration Register |
| MPAMF_BASE_rl | 0x0500 | MPAMCFG_MBW_PROP | MPAM Memory Bandwidth Proportional Stride Partition Configuration Register |
| MPAMF_BASE_rl | 0x0600 | MPAMCFG_INTPARTID | MPAM Internal PARTID Narrowing Configuration Register |
| MPAMF_BASE_rl | 0x0800 | MSMON_CFG_MON_SEL | MPAM Monitor Instance Selection Register |
| MPAMF_BASE_rl | 0x0808 | MSMON_CAPT_EVNT | MPAM Capture Event Generation Register |
| MPAMF_BASE_rl | 0x0810 | MSMON_CFG_CSU_FLT | MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register |
| MPAMF_BASE_rl | 0x0818 | MSMON_CFG_CSU_CTL | MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register |
| MPAMF_BASE_rl | 0x0820 | MSMON_CFG_MBWU_FLT | MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register |
| MPAMF_BASE_rl | 0x0828 | MSMON_CFG_MBWU_CTL | MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register |

| Frame | Offset | Name | Description |
|---------------|------------------|--|---|
| MPAMF_BASE_rl | 0x0840 | MSMON_CSU | MPAM Cache Storage Usage Monitor Register |
| MPAMF_BASE_rl | 0x0848 | MSMON_CSU_CAPTURE | MPAM Cache Storage Usage Monitor Capture Register |
| MPAMF_BASE_rl | 0x0858 | MSMON_CSU_OFSR | MPAM CSU Monitor Overflow Status Register |
| MPAMF_BASE_rl | 0x0860 | MSMON_MBWU | MPAM Memory Bandwidth Usage Monitor Register |
| MPAMF_BASE_rl | 0x0868 | MSMON_MBWU_CAPTURE | MPAM Memory Bandwidth Usage Monitor Capture Register |
| MPAMF_BASE_rl | 0x0880 | MSMON_MBWU_L | MPAM Long Memory Bandwidth Usage Monitor Register |
| MPAMF_BASE_rl | 0x0890 | MSMON_MBWU_L_CAPTURE | MPAM Long Memory Bandwidth Usage Monitor Capture Register |
| MPAMF_BASE_rl | 0x0898 | MSMON_MBWU_OFSR | MPAM MBWU Monitor Overflow Status Register |
| MPAMF_BASE_rl | 0x08DC | MSMON_OFLOW_MSI_MPAM | MPAM Monitor Overflow MSI Write MPAM Information Register |
| MPAMF_BASE_rl | 0x08E0 | MSMON_OFLOW_MSI_ADDR_L | MPAM Monitor Overflow MSI Low-part Address Register |
| MPAMF_BASE_rl | 0x08E4 | MSMON_OFLOW_MSI_ADDR_H | MPAM Monitor Overflow MSI Write High-part Address Register |
| MPAMF_BASE_rl | 0x08E8 | MSMON_OFLOW_MSI_DATA | MPAM Monitor Overflow MSI Write Data Register |
| MPAMF_BASE_rl | 0x08EC | MSMON_OFLOW_MSI_ATTR | MPAM Monitor Overflow MSI Write Attributes Register |
| MPAMF_BASE_rl | 0x08F0 | MSMON_OFLOW_SR | MPAM Monitor Overflow Status Register |
| MPAMF_BASE_rl | 0x1000 + (4 * n) | MPAMCFG_CPBM<n> | MPAM Cache Portion Bitmap Partition Configuration Register |
| MPAMF_BASE_rl | 0x2000 + (4 * n) | MPAMCFG_MBW_PBM<n> | MPAM Bandwidth Portion Bitmap Partition Configuration Register |
| MPAMF_BASE_rt | 0x0000 | MPAMF_IDR | MPAM Features Identification Register |
| MPAMF_BASE_rt | 0x0018 | MPAMF_IIDR | MPAM Implementation Identification Register |
| MPAMF_BASE_rt | 0x0020 | MPAMF_AIDR | MPAM Architecture Identification Register |
| MPAMF_BASE_rt | 0x0028 | MPAMF_IMPL_IDR | MPAM Implementation-Specific Partitioning Feature Identification Register |
| MPAMF_BASE_rt | 0x0030 | MPAMF_CPOR_IDR | MPAM Features Cache Portion Partitioning ID register |
| MPAMF_BASE_rt | 0x0038 | MPAMF_CCAP_IDR | MPAM Features Cache Capacity Partitioning ID register |
| MPAMF_BASE_rt | 0x0040 | MPAMF_MBW_IDR | MPAM Memory Bandwidth Partitioning Identification Register |

| Frame | Offset | Name | Description |
|---------------|--------|--------------------------------------|--|
| MPAMF_BASE_rt | 0x0048 | MPAMF_PRI_IDR | MPAM Priority Partitioning Identification Register |
| MPAMF_BASE_rt | 0x0050 | MPAMF_PARTID_NRW_IDR | MPAM PARTID Narrowing ID register |
| MPAMF_BASE_rt | 0x0080 | MPAMF_MSMON_IDR | MPAM Resource Monitoring Identification Register |
| MPAMF_BASE_rt | 0x0088 | MPAMF_CSUMON_IDR | MPAM Features Cache Storage Usage Monitoring ID register |
| MPAMF_BASE_rt | 0x0090 | MPAMF_MBWUMON_IDR | MPAM Features Memory Bandwidth Usage Monitoring ID register |
| MPAMF_BASE_rt | 0x00DC | MPAMF_ERR_MSI_MPAM | MPAM Error MSI Write MPAM Information Register |
| MPAMF_BASE_rt | 0x00E0 | MPAMF_ERR_MSI_ADDR_L | MPAM Error MSI Low-part Address Register |
| MPAMF_BASE_rt | 0x00E4 | MPAMF_ERR_MSI_ADDR_H | MPAM Error MSI High-part Address Register |
| MPAMF_BASE_rt | 0x00E8 | MPAMF_ERR_MSI_DATA | MPAM Error MSI Data Register |
| MPAMF_BASE_rt | 0x00EC | MPAMF_ERR_MSI_ATTR | MPAM Error MSI Write Attributes Register |
| MPAMF_BASE_rt | 0x00F0 | MPAMF_ECR | MPAM Error Control Register |
| MPAMF_BASE_rt | 0x00F8 | MPAMF_ESR | MPAM Error Status Register |
| MPAMF_BASE_rt | 0x0100 | MPAMCFG_PART_SEL | MPAM Partition Configuration Selection Register |
| MPAMF_BASE_rt | 0x0108 | MPAMCFG_CMAX | MPAM Cache Maximum Capacity Partition Configuration Register |
| MPAMF_BASE_rt | 0x0110 | MPAMCFG_CMIN | MPAM Cache Minimum Capacity Partition Configuration Register |
| MPAMF_BASE_rt | 0x0118 | MPAMCFG_CASSOC | MPAM Cache Maximum Associativity Partition Configuration Register |
| MPAMF_BASE_rt | 0x0200 | MPAMCFG_MBW_MIN | MPAM Memory Bandwidth Minimum Partition Configuration Register |
| MPAMF_BASE_rt | 0x0208 | MPAMCFG_MBW_MAX | MPAM Memory Bandwidth Maximum Partition Configuration Register |
| MPAMF_BASE_rt | 0x0220 | MPAMCFG_MBW_WINWD | MPAM Memory Bandwidth Partitioning Window Width Configuration Register |
| MPAMF_BASE_rt | 0x0300 | MPAMCFG_EN | MPAM Partition Configuration Enable Register |
| MPAMF_BASE_rt | 0x0310 | MPAMCFG_DIS | MPAM Partition Configuration Disable Register |
| MPAMF_BASE_rt | 0x0320 | MPAMCFG_EN_FLAGS | MPAM Partition Configuration Enable Flags Register |
| MPAMF_BASE_rt | 0x0400 | MPAMCFG_PRI | MPAM Priority Partition Configuration Register |

| Frame | Offset | Name | Description |
|---------------|--------|--|--|
| MPAMF_BASE_rt | 0x0500 | MPAMCFG_MBW_PROP | MPAM Memory Bandwidth Proportional Stride Partition Configuration Register |
| MPAMF_BASE_rt | 0x0600 | MPAMCFG_INTPARTID | MPAM Internal PARTID Narrowing Configuration Register |
| MPAMF_BASE_rt | 0x0800 | MSMON_CFG_MON_SEL | MPAM Monitor Instance Selection Register |
| MPAMF_BASE_rt | 0x0808 | MSMON_CAPT_EVNT | MPAM Capture Event Generation Register |
| MPAMF_BASE_rt | 0x0810 | MSMON_CFG_CSU_FLT | MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register |
| MPAMF_BASE_rt | 0x0818 | MSMON_CFG_CSU_CTL | MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register |
| MPAMF_BASE_rt | 0x0820 | MSMON_CFG_MBWU_FLT | MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register |
| MPAMF_BASE_rt | 0x0828 | MSMON_CFG_MBWU_CTL | MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register |
| MPAMF_BASE_rt | 0x0840 | MSMON_CSU | MPAM Cache Storage Usage Monitor Register |
| MPAMF_BASE_rt | 0x0848 | MSMON_CSU_CAPTURE | MPAM Cache Storage Usage Monitor Capture Register |
| MPAMF_BASE_rt | 0x0858 | MSMON_CSU_OFSR | MPAM CSU Monitor Overflow Status Register |
| MPAMF_BASE_rt | 0x0860 | MSMON_MBWU | MPAM Memory Bandwidth Usage Monitor Register |
| MPAMF_BASE_rt | 0x0868 | MSMON_MBWU_CAPTURE | MPAM Memory Bandwidth Usage Monitor Capture Register |
| MPAMF_BASE_rt | 0x0880 | MSMON_MBWU_L | MPAM Long Memory Bandwidth Usage Monitor Register |
| MPAMF_BASE_rt | 0x0890 | MSMON_MBWU_L_CAPTURE | MPAM Long Memory Bandwidth Usage Monitor Capture Register |
| MPAMF_BASE_rt | 0x0898 | MSMON_MBWU_OFSR | MPAM MBWU Monitor Overflow Status Register |
| MPAMF_BASE_rt | 0x08DC | MSMON_OFLOW_MSI_MPAM | MPAM Monitor Overflow MSI Write MPAM Information Register |
| MPAMF_BASE_rt | 0x08E0 | MSMON_OFLOW_MSI_ADDR_L | MPAM Monitor Overflow MSI Low-part Address Register |
| MPAMF_BASE_rt | 0x08E4 | MSMON_OFLOW_MSI_ADDR_H | MPAM Monitor Overflow MSI Write High-part Address Register |
| MPAMF_BASE_rt | 0x08E8 | MSMON_OFLOW_MSI_DATA | MPAM Monitor Overflow MSI Write Data Register |
| MPAMF_BASE_rt | 0x08EC | MSMON_OFLOW_MSI_ATTR | MPAM Monitor Overflow MSI Write Attributes Register |

| Frame | Offset | Name | Description |
|---------------|------------------|--|---|
| MPAMF_BASE_rt | 0x08F0 | MSMON_OFLOW_SR | MPAM Monitor Overflow Status Register |
| MPAMF_BASE_rt | 0x1000 + (4 * n) | MPAMCFG_CPB<n> | MPAM Cache Portion Bitmap Partition Configuration Register |
| MPAMF_BASE_rt | 0x2000 + (4 * n) | MPAMCFG_MBW_PBM<n> | MPAM Bandwidth Portion Bitmap Partition Configuration Register |
| MPAMF_BASE_s | 0x0000 | MPAMF_IDR | MPAM Features Identification Register |
| MPAMF_BASE_s | 0x0008 | MPAMF_SIDR | MPAM Features Secure Identification Register |
| MPAMF_BASE_s | 0x0018 | MPAMF_IIDR | MPAM Implementation Identification Register |
| MPAMF_BASE_s | 0x0020 | MPAMF_AIDR | MPAM Architecture Identification Register |
| MPAMF_BASE_s | 0x0028 | MPAMF_IMPL_IDR | MPAM Implementation-Specific Partitioning Feature Identification Register |
| MPAMF_BASE_s | 0x0030 | MPAMF_CPOR_IDR | MPAM Features Cache Portion Partitioning ID register |
| MPAMF_BASE_s | 0x0038 | MPAMF_CCAP_IDR | MPAM Features Cache Capacity Partitioning ID register |
| MPAMF_BASE_s | 0x0040 | MPAMF_MBW_IDR | MPAM Memory Bandwidth Partitioning Identification Register |
| MPAMF_BASE_s | 0x0048 | MPAMF_PRI_IDR | MPAM Priority Partitioning Identification Register |
| MPAMF_BASE_s | 0x0050 | MPAMF_PARTID_NRW_IDR | MPAM PARTID Narrowing ID register |
| MPAMF_BASE_s | 0x0080 | MPAMF_MSMON_IDR | MPAM Resource Monitoring Identification Register |
| MPAMF_BASE_s | 0x0088 | MPAMF_CSUMON_IDR | MPAM Features Cache Storage Usage Monitoring ID register |
| MPAMF_BASE_s | 0x0090 | MPAMF_MBWUMON_IDR | MPAM Features Memory Bandwidth Usage Monitoring ID register |
| MPAMF_BASE_s | 0x00DC | MPAMF_ERR_MSI_MPAM | MPAM Error MSI Write MPAM Information Register |
| MPAMF_BASE_s | 0x00E0 | MPAMF_ERR_MSI_ADDR_L | MPAM Error MSI Low-part Address Register |
| MPAMF_BASE_s | 0x00E4 | MPAMF_ERR_MSI_ADDR_H | MPAM Error MSI High-part Address Register |
| MPAMF_BASE_s | 0x00E8 | MPAMF_ERR_MSI_DATA | MPAM Error MSI Data Register |
| MPAMF_BASE_s | 0x00EC | MPAMF_ERR_MSI_ATTR | MPAM Error MSI Write Attributes Register |
| MPAMF_BASE_s | 0x00F0 | MPAMF_ECR | MPAM Error Control Register |
| MPAMF_BASE_s | 0x00F8 | MPAMF_ESR | MPAM Error Status Register |
| MPAMF_BASE_s | 0x0100 | MPAMCFG_PART_SEL | MPAM Partition Configuration Selection Register |

| Frame | Offset | Name | Description |
|--------------|--------|------------------------------------|--|
| MPAMF_BASE_s | 0x0108 | MPAMCFG_CMAX | MPAM Cache Maximum Capacity Partition Configuration Register |
| MPAMF_BASE_s | 0x0110 | MPAMCFG_CMIN | MPAM Cache Minimum Capacity Partition Configuration Register |
| MPAMF_BASE_s | 0x0118 | MPAMCFG_CASSOC | MPAM Cache Maximum Associativity Partition Configuration Register |
| MPAMF_BASE_s | 0x0200 | MPAMCFG_MBW_MIN | MPAM Memory Bandwidth Minimum Partition Configuration Register |
| MPAMF_BASE_s | 0x0208 | MPAMCFG_MBW_MAX | MPAM Memory Bandwidth Maximum Partition Configuration Register |
| MPAMF_BASE_s | 0x0220 | MPAMCFG_MBW_WINWD | MPAM Memory Bandwidth Partitioning Window Width Configuration Register |
| MPAMF_BASE_s | 0x0300 | MPAMCFG_EN | MPAM Partition Configuration Enable Register |
| MPAMF_BASE_s | 0x0310 | MPAMCFG_DIS | MPAM Partition Configuration Disable Register |
| MPAMF_BASE_s | 0x0320 | MPAMCFG_EN_FLAGS | MPAM Partition Configuration Enable Flags Register |
| MPAMF_BASE_s | 0x0400 | MPAMCFG_PRI | MPAM Priority Partition Configuration Register |
| MPAMF_BASE_s | 0x0500 | MPAMCFG_MBW_PROP | MPAM Memory Bandwidth Proportional Stride Partition Configuration Register |
| MPAMF_BASE_s | 0x0600 | MPAMCFG_INTPARTID | MPAM Internal PARTID Narrowing Configuration Register |
| MPAMF_BASE_s | 0x0800 | MSMON_CFG_MON_SEL | MPAM Monitor Instance Selection Register |
| MPAMF_BASE_s | 0x0808 | MSMON_CAPT_EVNT | MPAM Capture Event Generation Register |
| MPAMF_BASE_s | 0x0810 | MSMON_CFG_CSU_FLT | MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register |
| MPAMF_BASE_s | 0x0818 | MSMON_CFG_CSU_CTL | MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register |
| MPAMF_BASE_s | 0x0820 | MSMON_CFG_MBWU_FLT | MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register |
| MPAMF_BASE_s | 0x0828 | MSMON_CFG_MBWU_CTL | MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register |
| MPAMF_BASE_s | 0x0840 | MSMON_CSU | MPAM Cache Storage Usage Monitor Register |
| MPAMF_BASE_s | 0x0848 | MSMON_CSU_CAPTURE | MPAM Cache Storage Usage Monitor Capture Register |

| Frame | Offset | Name | Description |
|--------------|------------------|--|--|
| MPAMF_BASE_s | 0x0858 | MSMON_CSU_OFSR | MPAM CSU Monitor Overflow Status Register |
| MPAMF_BASE_s | 0x0860 | MSMON_MBWU | MPAM Memory Bandwidth Usage Monitor Register |
| MPAMF_BASE_s | 0x0868 | MSMON_MBWU_CAPTURE | MPAM Memory Bandwidth Usage Monitor Capture Register |
| MPAMF_BASE_s | 0x0880 | MSMON_MBWU_L | MPAM Long Memory Bandwidth Usage Monitor Register |
| MPAMF_BASE_s | 0x0890 | MSMON_MBWU_L_CAPTURE | MPAM Long Memory Bandwidth Usage Monitor Capture Register |
| MPAMF_BASE_s | 0x0898 | MSMON_MBWU_OFSR | MPAM MBWU Monitor Overflow Status Register |
| MPAMF_BASE_s | 0x08DC | MSMON_OFLOW_MSI_MPAM | MPAM Monitor Overflow MSI Write MPAM Information Register |
| MPAMF_BASE_s | 0x08E0 | MSMON_OFLOW_MSI_ADDR_L | MPAM Monitor Overflow MSI Low-part Address Register |
| MPAMF_BASE_s | 0x08E4 | MSMON_OFLOW_MSI_ADDR_H | MPAM Monitor Overflow MSI Write High-part Address Register |
| MPAMF_BASE_s | 0x08E8 | MSMON_OFLOW_MSI_DATA | MPAM Monitor Overflow MSI Write Data Register |
| MPAMF_BASE_s | 0x08EC | MSMON_OFLOW_MSI_ATTR | MPAM Monitor Overflow MSI Write Attributes Register |
| MPAMF_BASE_s | 0x08F0 | MSMON_OFLOW_SR | MPAM Monitor Overflow Status Register |
| MPAMF_BASE_s | 0x1000 + (4 * n) | MPAMCFG_CPBm<n> | MPAM Cache Portion Bitmap Partition Configuration Register |
| MPAMF_BASE_s | 0x2000 + (4 * n) | MPAMCFG_MBW_PBM<n> | MPAM Bandwidth Portion Bitmap Partition Configuration Register |

In the RAS block:

| Offset | Name | Description |
|------------------|------------------------------------|---|
| 0x000 + (64 * n) | ERR<n>FR | Error Record Feature Register |
| 0x008 + (64 * n) | ERR<n>CTRL | Error Record Control Register |
| 0x010 + (64 * n) | ERR<n>STATUS | Error Record Primary Status Register |
| 0x018 + (64 * n) | ERR<n>ADDR | Error Record Address Register |
| 0x020 + (64 * n) | ERR<n>MISC0 | Error Record Miscellaneous Register 0 |
| 0x028 + (64 * n) | ERR<n>MISC1 | Error Record Miscellaneous Register 1 |
| 0x030 + (64 * n) | ERR<n>MISC2 | Error Record Miscellaneous Register 2 |
| 0x038 + (64 * n) | ERR<n>MISC3 | Error Record Miscellaneous Register 3 |
| 0x800 + (64 * n) | ERR<n>PFGF | Pseudo-fault Generation Feature Register |
| 0x800 + (8 * n) | ERRIMPDEF<n> | IMPLEMENTATION DEFINED Register <n>; |
| 0x808 + (64 * n) | ERR<n>PFGCTL | Pseudo-fault Generation Control Register |
| 0x810 + (64 * n) | ERR<n>PFGCDN | Pseudo-fault Generation Countdown Register |
| 0xE00 | ERRGSR | Error Group Status Register |
| 0xE10 | ERRIIDR | Implementation Identification Register |
| 0xE80 | ERRFHICR0 | Fault Handling Interrupt Configuration Register 0 |

| Offset | Name | Description |
|-----------------|-----------------------------------|---|
| 0xE80 + (8 * n) | ERRIRQCR<n> | Generic Error Interrupt Configuration Register |
| 0xE88 | ERRFHICR1 | Fault Handling Interrupt Configuration Register 1 |
| 0xE8C | ERRFHICR2 | Fault Handling Interrupt Configuration Register 2 |
| 0xE90 | ERRERICR0 | Error Recovery Interrupt Configuration Register 0 |
| 0xE98 | ERRERICR1 | Error Recovery Interrupt Configuration Register 1 |
| 0xE9C | ERRERICR2 | Error Recovery Interrupt Configuration Register 2 |
| 0xEA0 | ERRCRICR0 | Critical Error Interrupt Configuration Register 0 |
| 0xEA8 | ERRCRICR1 | Critical Error Interrupt Configuration Register 1 |
| 0xEAC | ERRCRICR2 | Critical Error Interrupt Configuration Register 2 |
| 0xEF8 | ERRIRQSR | Error Interrupt Status Register |
| 0xFA8 | ERRDEVAFF | Device Affinity Register |
| 0xFBC | ERRDEVARCH | Device Architecture Register |
| 0xFC8 | ERRDEVID | Device Configuration Register |
| 0xFD0 | ERRPIDR4 | Peripheral Identification Register 4 |
| 0xFE0 | ERRPIDR0 | Peripheral Identification Register 0 |
| 0xFE4 | ERRPIDR1 | Peripheral Identification Register 1 |
| 0xFE8 | ERRPIDR2 | Peripheral Identification Register 2 |
| 0xFEC | ERRPIDR3 | Peripheral Identification Register 3 |
| 0xFF0 | ERRCIDR0 | Component Identification Register 0 |
| 0xFF4 | ERRCIDR1 | Component Identification Register 1 |
| 0xFF8 | ERRCIDR2 | Component Identification Register 2 |
| 0xFFC | ERRCIDR3 | Component Identification Register 3 |

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCFGR, Activity Monitors Configuration Register

The AMCFGR characteristics are:

Purpose

Global configuration register for the activity monitors.

Provides information on supported features, the number of counter groups implemented, the total number of activity monitor event counters implemented, and the size of the counters. AMCFGR is applicable to both the architected and the auxiliary counter groups.

Configuration

External register AMCFGR bits [31:0] are architecturally mapped to AArch64 System register [AMCFGR_EL0\[31:0\]](#).

External register AMCFGR bits [31:0] are architecturally mapped to AArch32 System register [AMCFGR\[31:0\]](#).

The power domain of AMCFGR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCFGR are RES0.

Attributes

AMCFGR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|----|----|------|----|----|------|-----|----|----|----|----|----|------|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NCG | | | | RES0 | | | HDBG | RAZ | | | | | | SIZE | | | | N | | | | | | | | | | | | | |

NCG, bits [31:28]

Defines the number of counter groups.

The number of implemented counter groups is [AMCFGR.NCG + 1].

If the number of implemented auxiliary activity monitor event counters is zero, this field has a value of 0b0000. Otherwise, this field has a value of 0b0001.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [27:25]

Reserved, RES0.

HDBG, bit [24]

Halt-on-debug supported.

This feature must be supported, and so this bit is 0b1.

| HDBG | Meaning |
|------|--|
| 0b0 | AMCR.HDBG is RES0. |
| 0b1 | AMCR.HDBG is read/write. |

Access to this field is **RO**.

Bits [23:14]

Reserved, RAZ.

SIZE, bits [13:8]

Defines the size of activity monitor event counters.

The size of the activity monitor event counters implemented by the Activity Monitors Extension is [AMCFGR.SIZE + 1].

The counters are 64-bit.

Note

Software also uses this field to determine the spacing of counters in the memory-map. The counters are at doubleword-aligned addresses.

Reads as 0b111111.

Access to this field is **RO**.

N, bits [7:0]

Defines the number of activity monitor event counters.

The total number of counters implemented in all groups by the Activity Monitors Extension is [AMCFGR.N + 1].

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMCFGR

AMCFGR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| AMU | 0xE00 | AMCFGR |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCGCR, Activity Monitors Counter Group Configuration Register

The AMCGCR characteristics are:

Purpose

Provides information on the number of activity monitor event counters implemented within each counter group.

Configuration

External register AMCGCR bits [31:0] are architecturally mapped to AArch64 System register [AMCGCR_EL0\[31:0\]](#).

External register AMCGCR bits [31:0] are architecturally mapped to AArch32 System register [AMCGCR\[31:0\]](#).

The power domain of AMCGCR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCGCR are RES0.

Attributes

AMCGCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|-------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | CG1NC | | | | | | | | CG0NC | | | | | | | |

Bits [31:16]

Reserved, RES0.

CG1NC, bits [15:8]

Counter Group 1 Number of Counters. The number of counters in the auxiliary counter group.

In an implementation that includes FEAT_AMUv1, the permitted range of values is 0 to 16.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CG0NC, bits [7:0]

Counter Group 0 Number of Counters. The number of counters in the architected counter group.

Reads as 0x04.

Access to this field is **RO**.

Accessing AMCGCR

AMCGCR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
|-----------|--------|----------|

AMCGCR, Activity Monitors Counter Group Configuration Register

| | | |
|-----|-------|--------|
| AMU | 0xCE0 | AMCGCR |
|-----|-------|--------|

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCIDR0, Activity Monitors Component Identification Register 0

The AMCIDR0 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Component identification scheme'.

Configuration

The power domain of AMCIDR0 is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when FEAT_AMUv1 is implemented.

Attributes

AMCIDR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL_0 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Preamble.

Reads as 0x0D.

Access to this field is **RO**.

Accessing AMCIDR0

AMCIDR0 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| AMU | 0xFF0 | AMCIDR0 |

Accesses on this interface are **RO**.

AMCIDR1, Activity Monitors Component Identification Register 1

The AMCIDR1 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Component identification scheme'.

Configuration

The power domain of AMCIDR1 is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when FEAT_AMUv1 is implemented.

Attributes

AMCIDR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | CLASS | | | PRMBL 1 | | | | |

Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

| CLASS | Meaning |
|--------|----------------------|
| 0b1001 | CoreSight component. |

Other values are defined by the CoreSight Architecture.

This field reads as 0x9.

PRMBL_1, bits [3:0]

Preamble.

Reads as 0b0000.

Access to this field is **RO**.

Accessing AMCIDR1

AMCIDR1 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| AMU | 0xFF4 | AMCIDR1 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCIDR2, Activity Monitors Component Identification Register 2

The AMCIDR2 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Component identification scheme'.

Configuration

The power domain of AMCIDR2 is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when FEAT_AMUv1 is implemented.

Attributes

AMCIDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL_2 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Preamble.

Reads as 0x05.

Access to this field is **RO**.

Accessing AMCIDR2

AMCIDR2 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| AMU | 0xFF8 | AMCIDR2 |

Accesses on this interface are **RO**.

AMCIDR3, Activity Monitors Component Identification Register 3

The AMCIDR3 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Component identification scheme'.

Configuration

The power domain of AMCIDR3 is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when FEAT_AMUv1 is implemented.

Attributes

AMCIDR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL_3 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Preamble.

Reads as 0xB1.

Access to this field is **RO**.

Accessing AMCIDR3

AMCIDR3 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| AMU | 0xFFC | AMCIDR3 |

Accesses on this interface are **RO**.

AMCNTENCLR0, Activity Monitors Count Enable Clear Register 0

The AMCNTENCLR0 characteristics are:

Purpose

Disable control bits for the architected activity monitors event counters, [AMEVCNTR0<n>](#).

Configuration

External register AMCNTENCLR0 bits [31:0] are architecturally mapped to AArch64 System register [AMCNTENCLR0_ELO\[31:0\]](#).

External register AMCNTENCLR0 bits [31:0] are architecturally mapped to AArch32 System register [AMCNTENCLR0\[31:0\]](#).

The power domain of AMCNTENCLR0 is IMPLEMENTATION DEFINED.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR0 are RES0.

Attributes

AMCNTENCLR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|----|----|----|----|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | RAZ/WI | | | | | | | | | | P3 | P2 | P1 | P0 | | |

Bits [31:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n>, bit [n], for n = 3 to 0

Activity monitor event counter disable bit for [AMEVCNTR0<n>](#).

Note

[AMCGCR.CG0NC](#) identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

| P<n> | Meaning |
|-------------------|---|
| 0b0 | When read, means that AMEVCNTR0<n> is disabled. |
| 0b1 | When read, means that AMEVCNTR0<n> is enabled. |

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMCNTENCLR0

AMCNTENCLR0 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|------------------|---------------|-----------------|
| AMU | 0xC20 | AMCNTENCLR0 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENCLR1, Activity Monitors Count Enable Clear Register 1

The AMCNTENCLR1 characteristics are:

Purpose

Disable control bits for the auxiliary activity monitors event counters, [AMEVCNTR1<n>](#).

Configuration

External register AMCNTENCLR1 bits [31:0] are architecturally mapped to AArch64 System register [AMCNTENCLR1_ELO\[31:0\]](#).

External register AMCNTENCLR1 bits [31:0] are architecturally mapped to AArch32 System register [AMCNTENCLR1\[31:0\]](#).

The power domain of AMCNTENCLR1 is IMPLEMENTATION DEFINED.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENCLR1 are RES0.

Attributes

AMCNTENCLR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

Bits [31:16]

Reserved, RES0.

P<n>, bit [n], for n = 15 to 0

Activity monitor event counter disable bit for [AMEVCNTR1<n>](#).

When N is less than 16, bits [15:N] are RAZ, where N is the value in [AMCGCR.CG1NC](#).

Possible values of each bit are:

| P<n> | Meaning |
|------|---|
| 0b0 | When read, means that AMEVCNTR1<n> is disabled. |
| 0b1 | When read, means that AMEVCNTR1<n> is enabled. |

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMCNTENCLR1

If the number of auxiliary activity monitor event counters implemented is zero, reads of AMCNTENCLR1 are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

The number of auxiliary activity monitor event counters implemented is zero exactly when [AMCFGR](#).NCG == 0b0000.

AMCNTENCLR1 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-------------|
| AMU | 0xC24 | AMCNTENCLR1 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENSET0, Activity Monitors Count Enable Set Register 0

The AMCNTENSET0 characteristics are:

Purpose

Enable control bits for the architected activity monitors event counters, [AMEVCNTR0<n>](#).

Configuration

External register AMCNTENSET0 bits [31:0] are architecturally mapped to AArch64 System register [AMCNTENSET0_ELO\[31:0\]](#).

External register AMCNTENSET0 bits [31:0] are architecturally mapped to AArch32 System register [AMCNTENSET0\[31:0\]](#).

The power domain of AMCNTENSET0 is IMPLEMENTATION DEFINED.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET0 are RES0.

Attributes

AMCNTENSET0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|----|----|----|----|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | RAZ/WI | | | | | | | | | | P3 | P2 | P1 | P0 | | |

Bits [31:16]

Reserved, RES0.

Bits [15:4]

Reserved, RAZ/WI.

This field is reserved for additional architected activity monitor event counters, which Arm might define in a future version of the Activity Monitors architecture.

P<n>, bit [n], for n = 3 to 0

Activity monitor event counter enable bit for [AMEVCNTR0<n>](#).

Note

[AMCGCR.CG0NC](#) identifies the number of architected activity monitor event counters. In an implementation that includes FEAT_AMUv1, the number of architected activity monitor event counters is 4.

Possible values of each bit are:

| P<n> | Meaning |
|-------------------|---|
| 0b0 | When read, means that AMEVCNTR0<n> is disabled. |
| 0b1 | When read, means that AMEVCNTR0<n> is enabled. |

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMCNTENSET0

AMCNTENSET0 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|------------------|---------------|-----------------|
| AMU | 0xC00 | AMCNTENSET0 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCNTENSET1, Activity Monitors Count Enable Set Register 1

The AMCNTENSET1 characteristics are:

Purpose

Enable control bits for the auxiliary activity monitors event counters, [AMEVCNTR1<n>](#).

Configuration

External register AMCNTENSET1 bits [31:0] are architecturally mapped to AArch64 System register [AMCNTENSET1_ELO\[31:0\]](#).

External register AMCNTENSET1 bits [31:0] are architecturally mapped to AArch32 System register [AMCNTENSET1\[31:0\]](#).

The power domain of AMCNTENSET1 is IMPLEMENTATION DEFINED.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCNTENSET1 are RES0.

Attributes

AMCNTENSET1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

Bits [31:16]

Reserved, RES0.

P<n>, bit [n], for n = 15 to 0

Activity monitor event counter enable bit for [AMEVCNTR1<n>](#).

When N is less than 16, bits [15:N] are RAZ, where N is the value in [AMCGCR.CG1NC](#).

Possible values of each bit are:

| P<n> | Meaning |
|------|---|
| 0b0 | When read, means that AMEVCNTR1<n> is disabled. |
| 0b1 | When read, means that AMEVCNTR1<n> is enabled. |

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMCNTENSET1

If the number of auxiliary activity monitor event counters implemented is zero, reads of AMCNTENSET1 are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

The number of auxiliary activity monitor counters implemented is zero exactly when [AMCFGR](#).NCG == 0b0000.

AMCNTENSET1 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-------------|
| AMU | 0xC04 | AMCNTENSET1 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMCR, Activity Monitors Control Register

The AMCR characteristics are:

Purpose

Global control register for the activity monitors implementation. AMCR is applicable to both the architected and the auxiliary counter groups.

Configuration

External register AMCR bits [31:0] are architecturally mapped to AArch64 System register [AMCR_EL0\[31:0\]](#).

External register AMCR bits [31:0] are architecturally mapped to AArch32 System register [AMCR\[31:0\]](#).

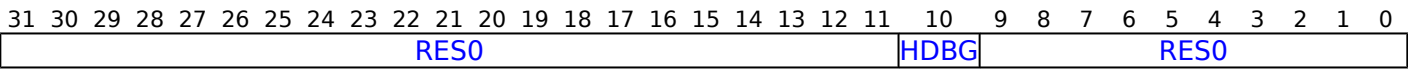
The power domain of AMCR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMCR are RES0.

Attributes

AMCR is a 32-bit register.

Field descriptions



Bits [31:11]

Reserved, RES0.

HDBG, bit [10]

This bit controls whether activity monitor counting is halted when the PE is halted in Debug state.

| HDBG | Meaning |
|------|--|
| 0b0 | Activity monitors do not halt counting when the PE is halted in Debug state. |
| 0b1 | Activity monitors halt counting when the PE is halted in Debug state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [9:0]

Reserved, RES0.

Accessing AMCR

AMCR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| AMU | 0xE04 | AMCR |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMDEVAFF0, Activity Monitors Device Affinity Register 0

The AMDEVAFF0 characteristics are:

Purpose

Copy of the low half of the PE [MPIDR_EL1](#) register that allows a debugger to determine which PE in a multiprocessor system the AMU component relates to.

Configuration

The power domain of AMDEVAFF0 is IMPLEMENTATION DEFINED.

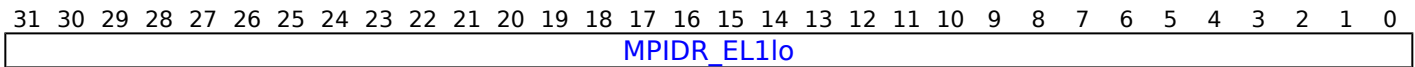
Implementation of this register is OPTIONAL.

This register is present only when FEAT_AMUv1 is implemented.

Attributes

AMDEVAFF0 is a 32-bit register.

Field descriptions



MPIDR_EL1lo, bits [31:0]

[MPIDR_EL1](#) low half. Read-only copy of the low half of [MPIDR_EL1](#), as seen from the highest implemented Exception level.

Accessing AMDEVAFF0

AMDEVAFF0 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-----------|
| AMU | 0xFA8 | AMDEVAFF0 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMDEVAFF1, Activity Monitors Device Affinity Register 1

The AMDEVAFF1 characteristics are:

Purpose

Copy of the high half of the PE [MPIDR_EL1](#) register that allows a debugger to determine which PE in a multiprocessor system the AMU component relates to.

Configuration

The power domain of AMDEVAFF1 is IMPLEMENTATION DEFINED.

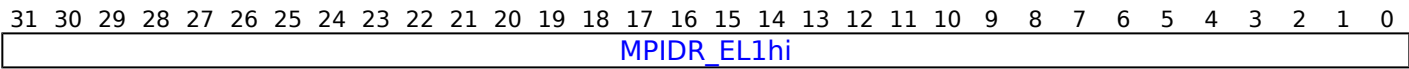
Implementation of this register is OPTIONAL.

This register is present only when FEAT_AMUv1 is implemented.

Attributes

AMDEVAFF1 is a 32-bit register.

Field descriptions



MPIDR_EL1hi, bits [31:0]

[MPIDR_EL1](#) high half. Read-only copy of the high half of [MPIDR_EL1](#), as seen from the highest implemented Exception level.

Accessing AMDEVAFF1

AMDEVAFF1 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-----------|
| AMU | 0xFAC | AMDEVAFF1 |

Accesses on this interface are **RO**.

AMDEVARCH, Activity Monitors Device Architecture Register

The AMDEVARCH characteristics are:

Purpose

Identifies the programmers' model architecture of the AMU component.

Configuration

The power domain of AMDEVARCH is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when FEAT_AMUv1 is implemented.

Attributes

AMDEVARCH is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|---------|----------|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ARCHITECT | | | | | | | | | | | PRESENT | REVISION | | | | ARCHID | | | | | | | | | | | | | | | |

ARCHITECT, bits [31:21]

Defines the architecture of the component. For AMU, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, 0x4.

Bits [27:21] are the JEP106 ID code, 0x3B.

Reads as 0b01000111011.

Access to this field is **RO**.

PRESENT, bit [20]

Indicates that the DEVARCH is present.

Reads as 0b1.

Access to this field is **RO**.

REVISION, bits [19:16]

Defines the architecture revision. For architectures defined by Arm this is the minor revision.

All other values are reserved.

Reads as 0b0000.

Access to this field is **RO**.

ARCHID, bits [15:0]

Defines this part to be an AMU component. For architectures defined by Arm this is further subdivided.

For AMU:

- Bits [15:12] are the architecture version, 0x0.
- Bits [11:0] are the architecture part number, 0xA66.

This corresponds to AMU architecture version AMUv1.

Reads as 0x0A66.

Access to this field is **RO**.

Accessing AMDEVARCH

AMDEVARCH can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-----------|
| AMU | 0xFBC | AMDEVARCH |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMDEVTTYPE, Activity Monitors Device Type Register

The AMDEVTTYPE characteristics are:

Purpose

Indicates to a debugger that this component is part of a PE's performance monitor interface.

Configuration

The power domain of AMDEVTTYPE is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when FEAT_AMUv1 is implemented.

Attributes

AMDEVTTYPE is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|-----|---|---|-------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | SUB | | | MAJOR | | | | |

Bits [31:8]

Reserved, RES0.

SUB, bits [7:4]

Subtype.

Reads as 0b0001.

Access to this field is **RO**.

MAJOR, bits [3:0]

Major type.

Reads as 0b0110.

Access to this field is **RO**.

Accessing AMDEVTTYPE

AMDEVTTYPE can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|------------|
| AMU | 0xFCC | AMDEVTTYPE |

Accesses on this interface are **RO**.

AMEVCNTR0<n>, Activity Monitors Event Counter Registers 0, n = 0 - 3

The AMEVCNTR0<n> characteristics are:

Purpose

Provides access to the architected activity monitor event counters.

Configuration

External register AMEVCNTR0<n> bits [63:0] are architecturally mapped to AArch64 System register [AMEVCNTR0<n>_ELO\[63:0\]](#).

External register AMEVCNTR0<n> bits [63:0] are architecturally mapped to AArch32 System register [AMEVCNTR0<n>\[63:0\]](#).

The power domain of AMEVCNTR0<n> is IMPLEMENTATION DEFINED.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR0<n> are RES0.

Attributes

AMEVCNTR0<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | ACNT | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ACNT | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ACNT, bits [63:0]

Architected activity monitor event counter n.

Value of architected activity monitor event counter n, where n is the number of this register and is a number from 0 to 3.

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMEVCNTR0<n>

If <n> is greater than or equal to the number of architected activity monitor event counters, reads of AMEVCNTR0<n> are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

[AMCGCR.CG0NC](#) identifies the number of architected activity monitor event counters.

AMEVCNTR0<n> can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance | Range |
|-----------|--------------------|--------------|-------|
| AMU | 0x000 + (8 * n) | AMEVCNTR0<n> | 31:0 |

Accesses on this interface are **RO**.

| Component | Offset | Instance | Range |
|-----------|--------------------|--------------|-------|
| AMU | 0x004 + (8 * n) | AMEVCNTR0<n> | 63:32 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVCNTR1<n>, Activity Monitors Event Counter Registers 1, n = 0 - 15

The AMEVCNTR1<n> characteristics are:

Purpose

Provides access to the auxiliary activity monitor event counters.

Configuration

External register AMEVCNTR1<n> bits [63:0] are architecturally mapped to AArch64 System register [AMEVCNTR1<n>_ELO\[63:0\]](#).

External register AMEVCNTR1<n> bits [63:0] are architecturally mapped to AArch32 System register [AMEVCNTR1<n>\[63:0\]](#).

The power domain of AMEVCNTR1<n> is IMPLEMENTATION DEFINED.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVCNTR1<n> are RES0.

Attributes

AMEVCNTR1<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | ACNT | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ACNT | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ACNT, bits [63:0]

Auxiliary activity monitor event counter n.

Value of auxiliary activity monitor event counter n, where n is the number of this register and is a number from 0 to 15.

The reset behavior of this field is:

- On an AMU reset, this field resets to 0.

Accessing AMEVCNTR1<n>

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads of AMEVCNTR1<n> are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

[AMCGCR.CG1NC](#) identifies the number of auxiliary activity monitor event counters.

AMEVCNTR1<n> can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance | Range |
|-----------|--------------------------|--------------|-------|
| AMU | $0 \times 100 + (8 * n)$ | AMEVCNTR1<n> | 31:0 |

Accesses on this interface are **RO**.

| Component | Offset | Instance | Range |
|-----------|--------------------------|--------------|-------|
| AMU | $0 \times 104 + (8 * n)$ | AMEVCNTR1<n> | 63:32 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVTYPER0<n>, Activity Monitors Event Type Registers 0, n = 0 - 3

The AMEVTYPER0<n> characteristics are:

Purpose

Provides information on the events that an architected activity monitor event counter [AMEVCNTR0<n>](#) counts.

Configuration

External register AMEVTYPER0<n> bits [31:0] are architecturally mapped to AArch64 System register [AMEVTYPER0<n>_EL0\[31:0\]](#).

External register AMEVTYPER0<n> bits [31:0] are architecturally mapped to AArch32 System register [AMEVTYPER0<n>\[31:0\]](#).

The power domain of AMEVTYPER0<n> is IMPLEMENTATION DEFINED.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER0<n> are RES0.

Attributes

AMEVTYPER0<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | evtCount | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the architected activity monitor event counter [AMEVCNTR0<n>](#). The value of this field is architecturally mandated for each architected counter.

The following table shows the mapping between required event numbers and the corresponding counters:

| evtCount | Meaning | Applies when |
|----------|----------------------------|--------------|
| 0x0011 | Processor frequency cycles | When n == 0 |
| 0x4004 | Constant frequency cycles | When n == 1 |
| 0x0008 | Instructions retired | When n == 2 |
| 0x4005 | Memory stall cycles | When n == 3 |

Accessing AMEVTYPER0<n>

If <n> is greater than or equal to the number of architected activity monitor event counters, reads of AMEVTYPER0<n> are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

[AMCGCR.CG0NC](#) identifies the number of architected activity monitor event counters.

AMEVTYPER0<n> can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|-------------------|---------------|
| AMU | $0x400 + (4 * n)$ | AMEVTYPER0<n> |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMEVTYPER1<n>, Activity Monitors Event Type Registers 1, n = 0 - 15

The AMEVTYPER1<n> characteristics are:

Purpose

Provides information on the events that an auxiliary activity monitor event counter [AMEVCNTR1<n>](#) counts.

Configuration

External register AMEVTYPER1<n> bits [31:0] are architecturally mapped to AArch64 System register [AMEVTYPER1<n>_EL0\[31:0\]](#).

External register AMEVTYPER1<n> bits [31:0] are architecturally mapped to AArch32 System register [AMEVTYPER1<n>\[31:0\]](#).

The power domain of AMEVTYPER1<n> is IMPLEMENTATION DEFINED.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMEVTYPER1<n> are RES0.

Attributes

AMEVTYPER1<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | evtCount | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

evtCount, bits [15:0]

Event to count. The event number of the event that is counted by the auxiliary activity monitor event counter [AMEVCNTR1<n>](#).

It is IMPLEMENTATION DEFINED what values are supported by each counter.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing AMEVTYPER1<n>

If <n> is greater than or equal to the number of auxiliary activity monitor event counters, reads of AMEVTYPER1<n> are RAZ. Software must treat reserved accesses as RES0. See 'Access requirements for reserved and unallocated registers'.

Note

[AMCGCR.CG1NC](#) identifies the number of auxiliary activity monitor event counters.

AMEVTYPER1<n> can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|-------------------|---------------|
| AMU | $0x480 + (4 * n)$ | AMEVTYPER1<n> |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMIIDR, Activity Monitors Implementation Identification Register

The AMIIDR characteristics are:

Purpose

Defines the implementer and revisions of the AMU.

Configuration

The power domain of AMIIDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_AMUv1 is implemented. Otherwise, direct accesses to AMIIDR are RES0.

Attributes

AMIIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----------|----|----|-------------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ProductID | | | | | | | | | | | | Variant | | | Revision | | | Implementer | | | | | | | | | | | | | |

ProductID, bits [31:20]

This field is an AMU part identifier.

If [AMPIDR0](#) is implemented, [AMPIDR0.PART_0](#) matches bits [27:20] of this field.

If [AMPIDR1](#) is implemented, [AMPIDR1.PART_1](#) matches bits [31:28] of this field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Variant, bits [19:16]

This field distinguishes product variants or major revisions of the product.

If [AMPIDR2](#) is implemented, [AMPIDR2.REVISION](#) matches AMIIDR.Variant.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

This field distinguishes minor revisions of the product.

If [AMPIDR3](#) is implemented, [AMPIDR3.REVAND](#) matches AMIIDR.Revision.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 code of the company that implemented the AMU.

For an Arm implementation, this field reads as 0x43B.

Bits [11:8] contain the JEP106 continuation code of the implementer.

Bit 7 is RES0

Bits [6:0] contain the JEP106 identity code of the implementer.

If [AMPIDR4](#) is implemented, [AMPIDR4](#).DES_2 matches bits [11:8] of this field.

If [AMPIDR2](#) is implemented, [AMPIDR2](#).DES_1 matches bits [6:4] of this field.

If [AMPIDR1](#) is implemented, [AMPIDR1](#).DES_0 matches bits [3:0] of this field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMIIDR

AMIIDR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| AMU | 0xE08 | AMIIDR |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMPIDR0, Activity Monitors Peripheral Identification Register 0

The AMPIDR0 characteristics are:

Purpose

Provides information to identify an activity monitors component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

The power domain of AMPIDR0 is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

This register is present only when FEAT_AMUv1 is implemented.

Attributes

AMPIDR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|--------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PART_0 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, least significant byte.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing AMPIDR0

AMPIDR0 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| AMU | 0xFE0 | AMPIDR0 |

Accesses on this interface are **RO**.

AMPIDR1, Activity Monitors Peripheral Identification Register 1

The AMPIDR1 characteristics are:

Purpose

Provides information to identify an activity monitors component.
For more information, see 'About the Peripheral identification scheme'.

Configuration

The power domain of AMPIDR1 is IMPLEMENTATION DEFINED.
Implementation of this register is OPTIONAL.
This register is present only when FEAT_AMUv1 is implemented.

Attributes

AMPIDR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---|--------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | DES_0 | | | | PART_1 | | | |

Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, least significant nibble of JEP106 ID code.
For Arm Limited, this field is 0b1011.
This field has an IMPLEMENTATION DEFINED value.
Access to this field is **RO**.

PART_1, bits [3:0]

Part number, most significant nibble.
This field has an IMPLEMENTATION DEFINED value.
Access to this field is **RO**.

Accessing AMPIDR1

AMPIDR1 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
|-----------|--------|----------|

| | | |
|-----|-------|---------|
| AMU | 0xFE4 | AMPIDR1 |
|-----|-------|---------|

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMPIDR2, Activity Monitors Peripheral Identification Register 2

The AMPIDR2 characteristics are:

Purpose

Provides information to identify an activity monitors component.
For more information, see 'About the Peripheral identification scheme'.

Configuration

The power domain of AMPIDR2 is IMPLEMENTATION DEFINED.
Implementation of this register is OPTIONAL.
This register is present only when FEAT_AMUv1 is implemented.

Attributes

AMPIDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|----------|---|---|-------|-------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | REVISION | | | JEDEC | DES_1 | | | |

Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Part major revision. Parts can also use this field to extend Part number to 16-bits.
This field has an IMPLEMENTATION DEFINED value.
Access to this field is **RO**.

JEDEC, bit [3]

Indicates a JEP106 identity code is used.
Reads as 0b1.
Access to this field is **RO**.

DES_1, bits [2:0]

Designer, most significant bits of JEP106 ID code.
For Arm Limited, this field is 0b011.
This field has an IMPLEMENTATION DEFINED value.
Access to this field is **RO**.

Accessing AMPIDR2

AMPIDR2 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| AMU | 0xFE8 | AMPIDR2 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMPIDR3, Activity Monitors Peripheral Identification Register 3

The AMPIDR3 characteristics are:

Purpose

Provides information to identify an activity monitors component.
For more information, see 'About the Peripheral identification scheme'.

Configuration

The power domain of AMPIDR3 is IMPLEMENTATION DEFINED.
Implementation of this register is OPTIONAL.
This register is present only when FEAT_AMUv1 is implemented.

Attributes

AMPIDR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|--------|---|---|---|------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | REVAND | | | | CMOD | | | |

Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Part minor revision. Parts using [AMPIDR2](#).REVISION as an extension to the Part number must use this field as a major revision number.
This field has an IMPLEMENTATION DEFINED value.
Access to this field is **RO**.

CMOD, bits [3:0]

Customer modified. Indicates someone other than the Designer has modified the component.
This field has an IMPLEMENTATION DEFINED value.
Access to this field is **RO**.

Accessing AMPIDR3

AMPIDR3 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| AMU | 0xFEC | AMPIDR3 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

AMPIDR4, Activity Monitors Peripheral Identification Register 4

The AMPIDR4 characteristics are:

Purpose

Provides information to identify an activity monitors component.
For more information, see 'About the Peripheral identification scheme'.

Configuration

The power domain of AMPIDR4 is IMPLEMENTATION DEFINED.
Implementation of this register is OPTIONAL.
This register is present only when FEAT_AMUv1 is implemented.

Attributes

AMPIDR4 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|------|---|---|---|-------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | SIZE | | | | DES_2 | | | |

Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

Size of the component. Log₂ of the number of 4KB pages from the start of the component to the end of the component ID registers.
Reads as 0b0000.
Access to this field is **RO**.

DES_2, bits [3:0]

Designer. JEP106 continuation code, least significant nibble.
For Arm Limited, this field is 0b0100.
This field has an IMPLEMENTATION DEFINED value.
Access to this field is **RO**.

Accessing AMPIDR4

AMPIDR4 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| AMU | 0xFD0 | AMPIDR4 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ASICCTL, CTI External Multiplexer Control register

The ASICCTL characteristics are:

Purpose

Can be used to provide IMPLEMENTATION DEFINED controls for the CTI. For example, the register might be used to control multiplexors for additional IMPLEMENTATION DEFINED triggers. The IMPLEMENTATION DEFINED controls provided by this register might modify the architecturally defined behavior of the CTI.

Note

The architecturally-defined triggers must not be multiplexed.

Configuration

It is IMPLEMENTATION DEFINED whether ASICCTL is implemented in the Core power domain or in the Debug power domain.

If it is implemented in the Core power domain then it is IMPLEMENTATION DEFINED whether it is in the Cold reset domain or the Warm reset domain.

This register must reset to a value that supports the architecturally-defined behavior of the CTI. Changing the value of the register from its reset value causes IMPLEMENTATION DEFINED behavior that might differ from the architecturally-defined behavior of the CTI.

Other than the requirements listed in this register description, all aspects of the reset behavior of the ASICCTL are IMPLEMENTATION DEFINED.

Attributes

ASICCTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ASICCTL

ASICCTL can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| CTI | 0x144 | ASICCTL |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register are **IMPDEF**.

CNTACR<n>, Counter-timer Access Control Registers, n = 0 - 7

The CNTACR<n> characteristics are:

Purpose

Provides top-level access controls for the elements of a timer frame. CNTACR<n> provides the controls for frame CNTBaseN.

In addition to the CNTACR<n> control:

- [CNTNSAR](#) controls whether CNTACR<n> is accessible by Non-secure accesses.
- If frame CNTEL0BaseN is implemented, the [CNTEL0ACR](#) in frame CNTBaseN provides additional control of accesses to frame CNTEL0BaseN.

Configuration

The power domain of CNTACR<n> is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Implemented only if the value of [CNTTIDR](#).Frame<n> is 1.

An implementation of the counters might not provide configurable access to some or all of the features. In this case, the associated field in the CNTACR<n> register is:

- RAZ/WI if access is always denied.
- RAO/WI if access is always permitted.

Attributes

CNTACR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|------|------|-------|------|------|------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | RWPT | RWVT | RVOFF | RFRQ | RVCT | RPCT |

Bits [31:6]

Reserved, RES0.

RWPT, bit [5]

Read/write access to the EL1 Physical Timer registers [CNTP_CVAL](#), [CNTP_TVAL](#), and [CNTP_CTL](#), in frame <n>.

| RWPT | Meaning |
|------|---|
| 0b0 | No access to the EL1 Physical Timer registers in frame <n>. The registers are RES0. |
| 0b1 | Read/write access to the EL1 Physical Timer registers in frame <n>. |

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

RWVT, bit [4]

Read/write access to the Virtual Timer register [CNTV_CVAL](#), [CNTV_TVAL](#), and [CNTV_CTL](#), in frame <n>.

| RWVT | Meaning |
|-------------|--|
| 0b0 | No access to the Virtual Timer registers in frame <n>. The registers are RES0. |
| 0b1 | Read/write access to the Virtual Timer registers in frame <n>. |

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

RVOFF, bit [3]

Read-only access to [CNTVOFF](#), in frame <n>.

| RVOFF | Meaning |
|--------------|--|
| 0b0 | No access to CNTVOFF in frame <n>. The register is RES0. |
| 0b1 | Read-only access to CNTVOFF in frame <n>. |

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

RFRQ, bit [2]

Read-only access to [CNTFRQ](#), in frame <n>.

| RFRQ | Meaning |
|-------------|---|
| 0b0 | No access to CNTFRQ in frame <n>. The register is RES0. |
| 0b1 | Read-only access to CNTFRQ in frame <n>. |

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

RVCT, bit [1]

Read-only access to [CNTVCT](#), in frame <n>.

| RVCT | Meaning |
|-------------|---|
| 0b0 | No access to CNTVCT in frame <n>. The register is RES0. |
| 0b1 | Read-only access to CNTVCT in frame <n>. |

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

RPCT, bit [0]

Read-only access to [CNTPCT](#), in frame <n>.

| RPCT | Meaning |
|-------------|---|
| 0b0 | No access to CNTPCT in frame <n>. The register is RES0. |
| 0b1 | Read-only access to CNTPCT in frame <n>. |

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTACR<n>

In a system that recognizes two Security states:

- CNTACR<n> is always accessible by Secure accesses.
- [CNTNSAR](#).NS<n> determines whether CNTACR<n> is accessible by Non-secure accesses.

CNTACR<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|------------|-----------------|-----------|
| Timer | CNTCTLBase | 0x040 + (4 * n) | CNTACR<n> |

Accesses on this interface are **RW**.

CNTCR, Counter Control Register

The CNTCR characteristics are:

Purpose

Enables the counter, controls the counter frequency setting, and controls counter behavior during debug.

Configuration

The power domain of CNTCR is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|------|---|---|------|------|----|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | FCREQ | | | | | | | RES0 | | | SCEN | HDBG | EN | | | | | |

Bits [31:18]

Reserved, RES0.

FCREQ, bits [17:8]

Frequency change request. Indicates the number of the entry in the Frequency modes table to select.

Selecting an unimplemented entry, or an entry that contains 0, has no effect on the counter.

The maximum number of entries in the Frequency modes table is IMPLEMENTATION DEFINED up to a maximum of 1004 entries, see 'The Frequency modes table'. An implementation is only required to implement an FCREQ field that can hold values from 0 to the highest supported Frequency modes table entry. Any unrequired most-significant bits of FCREQ can be implemented as RES0.

The reset behavior of this field is:

- On a Timer reset, this field resets to 0.

Bits [7:3]

Reserved, RES0.

SCEN, bit [2]

When FEAT_CNTSC is implemented:

Scale Enable.

| SCEN | Meaning |
|------|--|
| 0b0 | Scaling is not enabled. The counter value is incremented by 0x1.00000000 for each counter tick. |
| 0b1 | Scaling is enabled. The counter is incremented by CNTSCR.ScaleVal for each counter tick. |

The SCEN bit can only be changed when the counter is disabled, when CNTCR.EN == 0.

If the value of CNTCR.SCEN changes when CNTCR.EN == 1 then:

- The counter value becomes UNKNOWN.
- The counter value remains UNKNOWN on future ticks of the clock.

When the [CNTCV](#) register in the CNTControlBase frame of the memory mapped counter module is written to, the accumulated fraction information is reset to zero.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HDBG, bit [1]

Halt-on-debug. Controls whether a Halt-on-debug signal halts the system counter:

| HDBG | Meaning |
|------|--|
| 0b0 | System counter ignores Halt-on-debug. |
| 0b1 | Asserted Halt-on-debug signal halts system counter update. |

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

EN, bit [0]

Enables the counter:

| EN | Meaning |
|-----|--------------------------|
| 0b0 | System counter disabled. |
| 0b1 | System counter enabled. |

The reset behavior of this field is:

- On a Timer reset, this field resets to 0.

Accessing CNTCR

In a system that supports Secure and Non-secure memory maps the CNTControlBase frame, that includes this register, is implemented only in the Secure memory map.

CNTCR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|----------------|--------|----------|
| Timer | CNTControlBase | 0x000 | CNTCR |

Accesses on this interface are **RW**.

CNTCV, Counter Count Value register

The CNTCV characteristics are:

Purpose

Indicates the current count value.

Configuration

The power domain of CNTCV is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTCV is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | CountValue | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | CountValue | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

CountValue, bits [63:0]

Indicates the counter value.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTCV

| Frame | Accessibility |
|----------------|---------------|
| CNTControlBase | RW |
| CNTReadBase | RO |

A write to CNTCV must be visible in the [CNTPCT](#) register of each running processor in a finite time.

For the instance of the register in the CNTControlBase frame:

- In a system that supports Secure and Non-secure memory maps, the CNTControlBase frame, and therefore this register instance, is implemented only in the Secure memory map.
- If the counter is enabled, the effect of writing to the register is UNKNOWN.

In an implementation that supports 64-bit atomic memory accesses, this register must be accessible using a 64-bit atomic access.

CNTCV can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance | Range |
|-----------|----------------|--------|----------|-------|
| Timer | CNTControlBase | 0x008 | CNTCV | 63:0 |

Accesses on this interface are **RW**.

CNTCV, Counter Count Value register

| Component | Frame | Offset | Instance | Range |
|-----------|-------------|--------|----------|-------|
| Timer | CNTReadBase | 0x000 | CNTCV | 63:0 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTELOACR, Counter-timer EL0 Access Control Register

The CNTELOACR characteristics are:

Purpose

An implementation of CNTELOACR in the frame at CNTBaseN controls whether the [CNTPTCT](#), [CNTVCT](#), [CNTFRQ](#), EL1 Physical Timer, and Virtual Timer registers are visible in the frame at CNTELOBaseN.

Configuration

The power domain of CNTELOACR is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTELOACR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---------|----|---|---|---|---------|---|---|---|---|------|---|--|--|--|----------|--|--|--|--|----------|--|--|--|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | |
| | | | | | | | | | | RES0 | | | | | | | | | | ELOPTEN | | | | | ELOVTEN | | | | | RES0 | | | | | ELOVCTEN | | | | | ELOPCTEN | | | | |

Bits [31:10]

Reserved, RES0.

ELOPTEN, bit [9]

Second view read/write access control for the EL1 Physical Timer registers. This bit controls whether the [CNTPT_CVAL](#), [CNTPT_TVAL](#), and [CNTPT_CTL](#) registers in the current CNTBaseN frame are also accessible in the corresponding CNTELOBaseN frame.

| ELOPTEN | Meaning |
|----------------|---|
| 0b0 | No access. Registers are RES0 in the second view. |
| 0b1 | Access permitted. If the registers are accessible in the current frame then they are accessible in the second view. |

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

ELOVTEN, bit [8]

Second view read/write access control for the Virtual Timer registers. This bit controls whether the [CNTVT_CVAL](#), [CNTVT_TVAL](#), and [CNTVT_CTL](#) registers in the current CNTBaseN frame are also accessible in the corresponding CNTELOBaseN frame.

| EL0VTEN | Meaning |
|---------|---|
| 0b0 | No access. Registers are RES0 in the second view. |
| 0b1 | Access permitted. If the registers are accessible in the current frame then they are accessible in the second view. |

The definition of this bit means that, if the Virtual Timer registers are not implemented in the current CNTBaseN frame, then the Virtual Timer register addresses are RES0 in the corresponding CNTEL0BaseN frame, regardless of the value of this bit.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Bits [7:2]

Reserved, RES0.

EL0VCTEN, bit [1]

Second view read access control for [CNTVCT](#) and [CNTERQ](#).

| EL0VCTEN | Meaning |
|----------|---|
| 0b0 | CNTVCT is not visible in the second view. If EL0PCTEN is set to 0, CNTERQ is not visible in the second view. |
| 0b1 | Access permitted. If CNTVCT and CNTERQ are visible in the current frame then they are visible in the second view. |

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

EL0PCTEN, bit [0]

Second view read access control for [CNTPCT](#) and [CNTERQ](#).

| EL0PCTEN | Meaning |
|----------|---|
| 0b0 | CNTPCT is not visible in the second view. If EL0VCTEN is set to 0, CNTERQ is not visible in the second view. |
| 0b1 | Access permitted. If CNTPCT and CNTERQ are visible in the current frame then they are visible in the second view. |

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTEL0ACR

CNTEL0ACR can be implemented in any implemented CNTBaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

If CNTEL0ACR is not implemented in an implemented CNTBaseN frame:

- The register location in that frame is RAZ/WI.

- If the corresponding CNTEL0BaseN frame is implemented, the registers [CNTERQ](#), [CNTP_CTL](#), [CNTP_CVAL](#), [CNTP_TVAL](#), [CNTPCT](#), [CNTV_CTL](#), [CNTV_CVAL](#), [CNTV_TVAL](#), and [CNTVCT](#) are not visible in that frame.

CNTEL0ACR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|----------|--------|-----------|
| Timer | CNTBaseN | 0x014 | CNTEL0ACR |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTFID0, Counter Frequency ID

The CNTFID0 characteristics are:

Purpose

Indicates the base frequency of the system counter.

Configuration

The power domain of CNTFID0 is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

The possible frequencies for the system counter are stored in the Frequency modes table as 32-bit words starting with the base frequency, CNTFID0. For more information, see 'The Frequency modes table'.

The final entry in the Frequency modes table must be followed by a 32-bit word of zero value, to mark the end of the table.

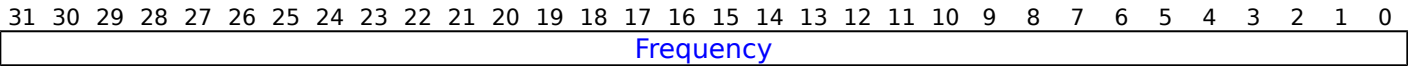
Typically, the Frequency modes table will be in read-only memory. However, a system implementation might use read/write memory for the table, and initialize the table entries as part of its start-up sequence.

If the Frequency modes table is in read/write memory, Arm strongly recommends that the table is not updated once the system is running.

Attributes

CNTFID0 is a 32-bit register.

Field descriptions



Frequency, bits [31:0]

- The base frequency of the system counter, in Hz.
- The reset behavior of this field is:
- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTFID0

It is IMPLEMENTATION DEFINED whether this register is RO or RW

In a system that supports Secure and Non-secure memory maps the CNTControlBase frame, that includes this register, is implemented only in the Secure memory map.

CNTFID0 can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|----------------|--------|----------|
| Timer | CNTControlBase | 0x020 | CNTFID0 |

Accesses on this interface are **RO or RW**.

CNTFID<n>, Counter Frequency IDs, n > 0, n = 1 - 1003

The CNTFID<n> characteristics are:

Purpose

Indicates alternative system counter update frequencies.

Configuration

The power domain of CNTFID<n> is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

The possible frequencies for the system counter are stored in the Frequency modes table as 32-bit words starting with the base frequency, [CNTFID0](#), see 'The Frequency modes table'.

The number of CNTFID<n> registers is IMPLEMENTATION DEFINED, and the only required CNTFID<n> register is [CNTFID0](#).

The final entry in the Frequency modes table must be followed by a 32-bit word of zero value, to mark the end of the table.

The architecture can support up to 1004 entries in the Frequency modes table, including the zero-word end marker, and the number of entries is IMPLEMENTATION DEFINED up to this limit. For an implementation that includes registers in the IMPLEMENTATION DEFINED register space 0x0C0-0x0FC, the maximum number of entries in the Frequency modes table is 40, including the zero-word end marker.

Typically, the Frequency modes table will be in read-only memory. However, a system implementation might use read/write memory for the table, and initialize the table entries as part of its start-up sequence.

If the Frequency modes table is in read/write memory, Arm strongly recommends that the table is not updated once the system is running.

Attributes

CNTFID<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------------------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | Frequency | | | | | | | | | | | | | | | |

Frequency, bits [31:0]

A system counter update frequency, in Hz. Must be an exact divisor of the base frequency. Arm strongly recommends that all frequency values in the Frequency modes table are integer power-of-two divisors of the base frequency.

When the system timer is operating at a lower frequency than the base frequency, the increment applied at each counter update is given by:

increment = (base frequency) / (selected frequency)

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTFID<n>

It is IMPLEMENTATION DEFINED whether this register is RO or RW

In a system that supports Secure and Non-secure memory maps the CNTControlBase frame, that includes these registers, is implemented only in the Secure memory map.

CNTFID<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|----------------|--------------------|-----------|
| Timer | CNTControlBase | 0x020 + (4 * n) | CNTFID<n> |

Accesses on this interface are **RO or RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTFRQ, Counter-timer Frequency

The CNTFRQ characteristics are:

Purpose

This register is provided so that software can discover the frequency of the system counter. The instance of the register in the CNTCTLBase frame must be programmed with this value as part of system initialization. The value of the register is not interpreted by hardware.

Configuration

The power domain of CNTFRQ is IMPLEMENTATION DEFINED.

For more information see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTFRQ is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Clock frequency | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Clock frequency. Indicates the system counter clock frequency, in Hz.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTFRQ

CNTFRQ must be implemented as an RW register in the CNTCTLBase frame.

In a system that recognizes two Security states, the instance of the register in the CNTCTLBase frame is only accessible by Secure accesses.

CNTFRQ can be implemented as a RO register in any implemented CNTBaseN frame, and in the corresponding CNTELOBaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTELOBaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTELOBaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTELOBaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame:

- CNTFRQ is accessible in that frame, as a RO register, if the value of [CNTACR<n>.RFRQ](#) is 1.
- Otherwise, the CNTFRQ address in that frame is RAZ/WI.

For an implemented CNTELOBaseN frame:

- CNTFRQ is accessible as a RO register in that frame if both:
 - CNTFRQ is accessible in the corresponding CNTBaseN frame.

- Either the value of [CNTELOACR.EL0VCTEN](#) is 1 or the value of [CNTELOACR.EL0PCTEN](#) is 1.
- Otherwise, the CNTFRQ address in that frame is RAZ/WI.

CNTFRQ can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|----------|--------|----------|
| Timer | CNTBaseN | 0x010 | CNTFRQ |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|-------------|--------|----------|
| Timer | CNTELOBaseN | 0x010 | CNTFRQ |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|------------|--------|----------|
| Timer | CNTCTLBase | 0x000 | CNTFRQ |

Accesses on this interface are **RO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTID, Counter Identification Register

The CNTID characteristics are:

Purpose

Indicates whether counter scaling is implemented.

Configuration

The power domain of CNTID is IMPLEMENTATION DEFINED.

This register is present only when FEAT_CNTSC is implemented. Otherwise, direct accesses to CNTID are RES0.

Attributes

CNTID is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | CNTSC | | | | | | | | | | | | | |

Bits [31:4]

Reserved, RES0.

CNTSC, bits [3:0]

Indicates whether Counter Scaling is implemented

| CNTSC | Meaning |
|--------|-------------------------------------|
| 0b0000 | Counter scaling is not implemented. |
| 0b0001 | Counter scaling is implemented. |

All other values are reserved.

Accessing CNTID

In a system that supports Secure and Non-secure memory maps, the CNTControlBase frame, that includes this register, is implemented only in the Secure memory map.

CNTID can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|----------------|--------|----------|
| Timer | CNTControlBase | 0x1C | CNTID |

Accesses on this interface are **RO**.

CNTNSAR, Counter-timer Non-secure Access Register

The CNTNSAR characteristics are:

Purpose

Provides the highest-level control of whether frames CNTBaseN and CNTELOBaseN are accessible by Non-secure accesses.

Configuration

The power domain of CNTNSAR is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTNSAR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-----|-----|-----|-----|-----|-----|-----|-----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | NS7 | NS6 | NS5 | NS4 | NS3 | NS2 | NS1 | NS0 |

Bits [31:8]

Reserved, RES0.

NS<n>, bit [n], for n = 7 to 0

Non-secure access to frame n.

| NS<n> | Meaning |
|-------|---|
| 0b0 | Secure access only. Behaves as RES0 to Non-secure accesses. |
| 0b1 | Secure and Non-secure accesses permitted. |

This bit also determines whether, in the CNTCTLBase frame, [CNTACR<n>](#) and [CNTVOFF<n>](#) are accessible to Non-secure accesses.

If frame CNTBase<n>:

- Is not implemented, then NS<n> is RES0.
- Is not Configurable access, and is accessible only by Secure accesses, then NS<n> is RES0.
- Is not Configurable access, and is accessible by both Secure and Non-secure accesses, then NS<n> is RES1.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTNSAR

In a system that recognizes two Security states, this register is only accessible by Secure accesses.

CNTNSAR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|------------|--------|----------|
| Timer | CNTCTLBase | 0x004 | CNTNSAR |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_CTL, Counter-timer Physical Timer Control

The CNTP_CTL characteristics are:

Purpose

Control register for the EL1 physical timer.

Configuration

The power domain of CNTP_CTL is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTP_CTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|-------|----|--------|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | ISTATUS | | | IMASK | | ENABLE | | | | | | | | | | |

Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTP_TVAL](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_CTL

CNTP_CTL can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame:

- CNTP_CTL is accessible in that frame if the value of [CNTACR<n>.RWPT](#) is 1.
- Otherwise, the CNTP_CTL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTP_CTL is accessible in that frame if both:
 - CNTP_CTL is accessible in the corresponding CNTBaseN frame:
 - The value of [CNTEL0ACR.EL0PTEN](#) is 1.
- Otherwise, the CNTP_CTL address in that frame is RAZ/WI.

CNTP_CTL can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|----------|--------|----------|
| Timer | CNTBaseN | 0x02C | CNTP_CTL |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|-------------|--------|----------|
| Timer | CNTEL0BaseN | 0x02C | CNTP_CTL |

Accesses on this interface are **RW**.

CNTP_CVAL, Counter-timer Physical Timer CompareValue

The CNTP_CVAL characteristics are:

Purpose

Holds the 64-bit compare value for the EL1 physical timer.

Configuration

The power domain of CNTP_CVAL is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTP_CVAL is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

CompareValue, bits [63:0]

Holds the EL1 physical timer CompareValue.

When [CNTP_CTL.ENABLE](#) is 1, the timer condition is met when ([CNTPCT](#) - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTP_CTL.ISTATUS](#) is set to 1.
- An interrupt is generated if [CNTP_CTL.IMASK](#) is 0.

When [CNTP_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTPCT](#) continues to count.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_CVAL

CNTP_CVAL can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTELOBaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTELOBaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTELOBaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTELOBaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame:

- CNTP_CVAL is accessible in that frame if the value of [CNTACR<n>.RWPT](#) is 1.

- Otherwise, the CNTP_CVAL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTP_CVAL is accessible in that frame if both:
 - CNTP_CVAL is accessible in the corresponding CNTBaseN frame:
 - The value of [CNTEL0ACR.EL0PTEN](#) is 1.
- Otherwise, the CNTP_CVAL address in that frame is RAZ/WI.

If the implementation supports 64-bit atomic accesses, then the CNTP_CVAL register must be accessible as an atomic 64-bit value.

CNTP_CVAL can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance | Range |
|-----------|----------|--------|-----------|-------|
| Timer | CNTBaseN | 0x020 | CNTP_CVAL | 31:0 |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance | Range |
|-----------|----------|--------|-----------|-------|
| Timer | CNTBaseN | 0x024 | CNTP_CVAL | 63:32 |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance | Range |
|-----------|-------------|--------|-----------|-------|
| Timer | CNTEL0BaseN | 0x020 | CNTP_CVAL | 31:0 |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance | Range |
|-----------|-------------|--------|-----------|-------|
| Timer | CNTEL0BaseN | 0x024 | CNTP_CVAL | 63:32 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTP_TVAL, Counter-timer Physical Timer TimerValue

The CNTP_TVAL characteristics are:

Purpose

Holds the timer value for the EL1 physical timer.

Configuration

The power domain of CNTP_TVAL is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTP_TVAL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | TimerValue | | | | | | | | | | | | | | | |

TimerValue, bits [31:0]

The TimerValue view of the EL1 physical timer.

On a read of this register:

- If [CNTP_CTL.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTP_CTL.ENABLE](#) is 1, the value returned is (CompareValue - [CNTPCT](#)).

On a write of this register, CompareValue is set to ([CNTPCT](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTP_CTL.ENABLE](#) is 1, the timer condition is met when ([CNTPCT](#) - CompareValue) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTP_CTL.ISTATUS](#) is set to 1.
- If [CNTP_CTL.IMASK](#) is 0, an interrupt is generated.

When [CNTP_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTPCT](#) continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTP_TVAL

CNTP_TVAL can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame:

- CNTP_TVAL is accessible in that frame if the value of [CNTACR<n>.RWPT](#) is 1.
- Otherwise, the CNTP_TVAL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTP_TVAL is accessible in that frame if both:
 - CNTP_TVAL is accessible in the corresponding CNTBaseN frame:
 - The value of [CNTEL0ACR.EL0PTEN](#) is 1.
- Otherwise, the CNTP_TVAL address in that frame is RAZ/WI.

CNTP_TVAL can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|----------|--------|-----------|
| Timer | CNTBaseN | 0x028 | CNTP_TVAL |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|-------------|--------|-----------|
| Timer | CNTEL0BaseN | 0x028 | CNTP_TVAL |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNPCT, Counter-timer Physical Count

The CNTPCT characteristics are:

Purpose

Holds the 64-bit physical count value.

Configuration

The power domain of CNTPCT is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNPCT is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Physical count value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Physical count value.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTPCT

CNPCT can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame, as a RO register.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame:

- CNPCT is accessible in that frame, as a RO register, if the value of [CNTACR<n>.RPCT](#) is 1.
- Otherwise, the CNTPCT address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNPCT is accessible in that frame if both:
 - CNPCT is accessible in the corresponding CNTBaseN frame.
 - The value of [CNTEL0ACR.EL0PCTEN](#) is 1.
- Otherwise, the CNTPCT address in that frame is RAZ/WI.

If the implementation supports 64-bit atomic accesses, then the CNTPCT register must be accessible as an atomic 64-bit value.

CNPCT can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance | Range |
|-----------|----------|--------|----------|-------|
| Timer | CNTBaseN | 0x000 | CNPCT | 31:0 |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance | Range |
|-----------|----------|--------|----------|-------|
| Timer | CNTBaseN | 0x004 | CNPCT | 63:32 |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance | Range |
|-----------|-------------|--------|----------|-------|
| Timer | CNTELOBaseN | 0x000 | CNPCT | 31:0 |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance | Range |
|-----------|-------------|--------|----------|-------|
| Timer | CNTELOBaseN | 0x004 | CNPCT | 63:32 |

Accesses on this interface are **RO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTSCR, Counter Scale Register

The CNTSCR characteristics are:

Purpose

Enables the counter, controls the counter frequency setting, and controls counter behavior during debug.

Configuration

The power domain of CNTSCR is IMPLEMENTATION DEFINED.

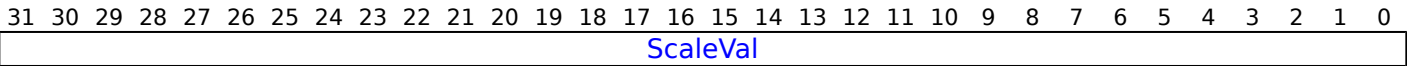
This register is present only when FEAT_CNTSC is implemented. Otherwise, direct accesses to CNTSCR are RES0.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTSCR is a 32-bit register.

Field descriptions



ScaleVal, bits [31:0]

Scale Value

When counter scaling is enabled, ScaleVal is the amount added to the counter value for every counter tick.

Counter tick is defined as one period of the current operating frequency of the Generic counter.

ScaleVal is expressed as an unsigned fixed point number with an 8-bit integer value and a 24-bit fractional value.

CNTSCR.ScaleVal can only be changed when [CNTCR.EN](#) == 0. If the value of this field is changed when [CNTCR.EN](#) == 1:

- The counter value becomes UNKNOWN.
- The counter value remains UNKNOWN on future ticks of the clock.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTSCR

In a system that supports Secure and Non-secure memory maps the CNTControlBase frame, that includes this register, is implemented only in the Secure memory map.

CNTSCR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|----------------|--------|----------|
| Timer | CNTControlBase | 0x10 | CNTSCR |

Accesses on this interface are **RW**.

CNTSR, Counter Status Register

The CNTSR characteristics are:

Purpose

Provides counter frequency status information.

Configuration

The power domain of CNTSR is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|------|---|---|---|------|------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | FACK | | | | | | | | RES0 | | | | DBGH | RES0 | | | | |

Bits [31:18]

Reserved, RES0.

FACK, bits [17:8]

Frequency Change Acknowledge. Indicates the currently selected entry in the Frequency modes table, see 'The Frequency modes table'.

The reset behavior of this field is:

- On a Timer reset, this field resets to 0.

Bits [7:2]

Reserved, RES0.

DBGH, bit [1]

Indicates whether the counter is halted because the Halt-on-debug signal is asserted:

| DBGH | Meaning |
|------|------------------------|
| 0b0 | Counter is not halted. |
| 0b1 | Counter is halted. |

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Bit [0]

Reserved, RES0.

Accessing CNTR

In a system that supports Secure and Non-secure memory maps the CNTControlBase frame, that includes this register, is implemented only in the Secure memory map.

CNTR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|----------------|--------|----------|
| Timer | CNTControlBase | 0x004 | CNTR |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTTIDR, Counter-timer Timer ID Register

The CNTTIDR characteristics are:

Purpose

Indicates the implemented timers in the memory map, and their features. For each value of N from 0 to 7 it indicates whether:

- Frame CNTBaseN is a view of an implemented timer.
- Frame CNTBaseN has a second view, CNTELOBaseN.
- Frame CNTBaseN has a virtual timer capability.

Configuration

The power domain of CNTTIDR is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTTIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|--------|----|----|----|--------|----|----|----|--------|----|----|----|--------|----|----|----|--------|----|---|---|--------|---|---|---|--------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Frame7 | | | | Frame6 | | | | Frame5 | | | | Frame4 | | | | Frame3 | | | | Frame2 | | | | Frame1 | | | | Frame0 | | | |

Frame<n>, bits [4n+3:4n], for n = 7 to 0

A 4-bit field indicating the features of frame CNTBase<n>.

Bit[3] of the field is RES0.

Bit[2], the FEL0 subfield, indicates whether frame CNTBase<n> has a second view, CNTELOBase<n>. The possible values of this bit are:

| Bit[2] | Meaning |
|--------|--|
| 0b0 | Frame<n> does not have a second view. The CNTELOACR register in the first view of the frame is RES0. |
| 0b1 | Frame<n> has a second view, CNTELOBase<n>. |

If bit[0] is 0, bit[2] is RES0.

Bit[1], the FVI subfield, indicates whether both:

- Frame CNTBase<n> implements the virtual timer registers [CNTV_CVAL](#), [CNTV_TVAL](#), and [CNTV_CTL](#).
- This CNTCTLBase frame implements the virtual timer offset register [CNTVOFF<n>](#).

The possible values of bit[1] are:

| Bit[1] | Meaning |
|--------|--|
| 0b0 | Frame<n> does not have virtual capability. The virtual time and offset registers are RES0. |
| 0b1 | Frame<n> has virtual capability. The virtual time and offset registers are implemented. |

If bit[0] is 0, bit[1] is RES0.

Bit[0], the FI subfield, indicates whether frame CNTBase<n> is implemented. The possible values of this bit are:

| Bit[0] | Meaning |
|--------|--|
| 0b0 | Frame<n> is not implemented. All registers associated with the frame are RES0. |
| 0b1 | Frame<n> is implemented |

Accessing CNTTIDR

In a system that recognizes two Security states this register is accessible by both Secure and Non-secure accesses.

CNTTIDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|------------|--------|----------|
| Timer | CNTCTLBase | 0x008 | CNTTIDR |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_CTL, Counter-timer Virtual Timer Control

The CNTV_CTL characteristics are:

Purpose

Control register for the virtual timer.

Configuration

The power domain of CNTV_CTL is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTV_CTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|-------|----|--------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | ISTATUS | | IMASK | | ENABLE | | | | | | | | | |

Bits [31:3]

Reserved, RES0.

ISTATUS, bit [2]

The status of the timer. This bit indicates whether the timer condition is met:

| ISTATUS | Meaning |
|---------|-----------------------------|
| 0b0 | Timer condition is not met. |
| 0b1 | Timer condition is met. |

When the value of the ENABLE bit is 1, ISTATUS indicates whether the timer condition is met. ISTATUS takes no account of the value of the IMASK bit. If the value of ISTATUS is 1 and the value of IMASK is 0 then the timer interrupt is asserted.

When the value of the ENABLE bit is 0, the ISTATUS field is UNKNOWN.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

IMASK, bit [1]

Timer interrupt mask bit. Permitted values are:

| IMASK | Meaning |
|-------|---|
| 0b0 | Timer interrupt is not masked by the IMASK bit. |
| 0b1 | Timer interrupt is masked by the IMASK bit. |

For more information, see the description of the ISTATUS bit.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

ENABLE, bit [0]

Enables the timer. Permitted values are:

| ENABLE | Meaning |
|--------|-----------------|
| 0b0 | Timer disabled. |
| 0b1 | Timer enabled. |

Setting this bit to 0 disables the timer output signal, but the timer value accessible from [CNTV_TVAL](#) continues to count down.

Note

Disabling the output signal might be a power-saving option.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_CTL

CNTV_CTL can be implemented in any implemented CNTBaseN frame that has virtual timer capability, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame that has virtual timer capability:

- CNTV_CTL is accessible in that frame if the value of [CNTACR<n>.RWVT](#) is 1.
- Otherwise, the CNTV_CTL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTV_CTL is accessible in that frame if both:
 - CNTV_CTL is accessible in the corresponding CNTBaseN frame:
 - The value of [CNTEL0ACR.EL0VTEN](#) is 1.
- Otherwise, the CNTV_CTL address in that frame is RAZ/WI.

CNTV_CTL can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|----------|--------|----------|
| Timer | CNTBaseN | 0x03C | CNTV_CTL |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|-------------|--------|----------|
| Timer | CNTEL0BaseN | 0x03C | CNTV_CTL |

Accesses on this interface are **RW**.

CNTV_CVAL, Counter-timer Virtual Timer CompareValue

The CNTV_CVAL characteristics are:

Purpose

Holds the 64-bit compare value for the virtual timer.

Configuration

The power domain of CNTV_CVAL is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTV_CVAL is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CompareValue | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

CompareValue, bits [63:0]

Holds the virtual timer CompareValue.

When [CNTV_CTL.ENABLE](#) is 1, the timer condition is met when ([CNTVCT](#) - CompareValue) is greater than or equal to zero. This means that CompareValue acts like a 64-bit upcounter timer. When the timer condition is met:

- [CNTV_CTL.ISTATUS](#) is set to 1.
- An interrupt is generated if [CNTV_CTL.IMASK](#) is 0.

When [CNTV_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTVCT](#) continues to count.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_CVAL

CNTV_CVAL can be implemented in any implemented CNTBaseN frame that has virtual timer capability, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame that has virtual timer capability:

- CNTV_CVAL is accessible in that frame if the value of [CNTACR<n>.RWVT](#) is 1.

- Otherwise, the CNTV_CVAL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTV_CVAL is accessible in that frame if both:
 - CNTV_CVAL is accessible in the corresponding CNTBaseN frame:
 - The value of [CNTEL0ACR.EL0VTEN](#) is 1.
- Otherwise, the CNTV_CVAL address in that frame is RAZ/WI.

If the implementation supports 64-bit atomic accesses, then the CNTV_CVAL register must be accessible as an atomic 64-bit value.

CNTV_CVAL can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance | Range |
|-----------|----------|--------|-----------|-------|
| Timer | CNTBaseN | 0x030 | CNTV_CVAL | 31:0 |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance | Range |
|-----------|----------|--------|-----------|-------|
| Timer | CNTBaseN | 0x034 | CNTV_CVAL | 63:32 |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance | Range |
|-----------|-------------|--------|-----------|-------|
| Timer | CNTEL0BaseN | 0x030 | CNTV_CVAL | 31:0 |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance | Range |
|-----------|-------------|--------|-----------|-------|
| Timer | CNTEL0BaseN | 0x034 | CNTV_CVAL | 63:32 |

Accesses on this interface are **RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTV_TVAL, Counter-timer Virtual Timer TimerValue

The CNTV_TVAL characteristics are:

Purpose

Holds the timer value for the virtual timer.

Configuration

The power domain of CNTV_TVAL is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTV_TVAL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | TimerValue | | | | | | | | | | | | | | | |

TimerValue, bits [31:0]

The TimerValue view of the virtual timer.

On a read of this register:

- If [CNTV_CTL.ENABLE](#) is 0, the value returned is UNKNOWN.
- If [CNTV_CTL.ENABLE](#) is 1, the value returned is (CompareValue - [CNTVCT](#)).

On a write of this register, CompareValue is set to ([CNTVCT](#) + TimerValue), where TimerValue is treated as a signed 32-bit integer.

When [CNTV_CTL.ENABLE](#) is 1, the timer condition is met when ([CNTVCT](#) - CompareValue) is greater than or equal to zero. This means that TimerValue acts like a 32-bit downcounter timer. When the timer condition is met:

- [CNTV_CTL.ISTATUS](#) is set to 1.
- If [CNTV_CTL.IMASK](#) is 0, an interrupt is generated.

When [CNTV_CTL.ENABLE](#) is 0, the timer condition is not met, but [CNTVCT](#) continues to count, so the TimerValue view appears to continue to count down.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTV_TVAL

CNTV_TVAL can be implemented in any implemented CNTBaseN frame that has virtual timer capability, and in the corresponding CNTEL0BaseN frame.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame that has virtual timer capability:

- CNTV_TVAL is accessible in that frame if the value of [CNTACR<n>](#).RWVT is 1.
- Otherwise, the CNTV_TVAL address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTV_TVAL is accessible in that frame if both:
 - CNTV_TVAL is accessible in the corresponding CNTBaseN frame:
 - The value of [CNTEL0ACR](#).EL0VTEN is 1.
- Otherwise, the CNTV_TVAL address in that frame is RAZ/WI.

CNTV_TVAL can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|----------|--------|-----------|
| Timer | CNTBaseN | 0x038 | CNTV_TVAL |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|-------------|--------|-----------|
| Timer | CNTEL0BaseN | 0x038 | CNTV_TVAL |

Accesses on this interface are **RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVCT, Counter-timer Virtual Count

The CNTVCT characteristics are:

Purpose

Holds the 64-bit virtual count value.

Configuration

The power domain of CNTVCT is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTVCT is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual count value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual count value.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVCT

CNTVCT can be implemented in any implemented CNTBaseN frame, and in the corresponding CNTEL0BaseN frame, as a RO register.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame:

- CNTVCT is accessible in that frame, as a RO register, if the value of [CNTACR<n>.RVCT](#) is 1.
- Otherwise, the CNTVCT address in that frame is RAZ/WI.

For an implemented CNTEL0BaseN frame:

- CNTVCT is accessible in that frame if both:
 - CNTVCT is accessible in the corresponding CNTBaseN frame.
 - The value of [CNTEL0ACR.EL0VCTEN](#) is 1.
- Otherwise, the CNTVCT address in that frame is RAZ/WI.

If the implementation supports 64-bit atomic accesses, then the CNTVCT register must be accessible as an atomic 64-bit value.

CNTVCT can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance | Range |
|-----------|----------|--------|----------|-------|
| Timer | CNTBaseN | 0x008 | CNTVCT | 31:0 |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance | Range |
|-----------|----------|--------|----------|-------|
| Timer | CNTBaseN | 0x00C | CNTVCT | 63:32 |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance | Range |
|-----------|-------------|--------|----------|-------|
| Timer | CNTELOBaseN | 0x008 | CNTVCT | 31:0 |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance | Range |
|-----------|-------------|--------|----------|-------|
| Timer | CNTELOBaseN | 0x00C | CNTVCT | 63:32 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVOFF, Counter-timer Virtual Offset

The CNTVOFF characteristics are:

Purpose

Holds the 64-bit virtual offset for a CNTBaseN frame that has virtual timer capability. This is the offset between real time and virtual time.

Configuration

The power domain of CNTVOFF is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTVOFF is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Virtual offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:0]

Virtual offset.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVOFF

CNTVOFF is implemented, as a RO register, in any implemented CNTBaseN frame that has virtual timer capability.

'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a CNTBaseN frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

For an implemented CNTBaseN frame that has virtual timer capability:

- CNTVOFF is accessible in that frame, as a RO register, if the value of [CNTACR<n>.RVOFF](#) is 1.
- Otherwise, the CNTVOFF address in that frame is RAZ/WI.

Note

CNTVOFF is never visible in any CNTEL0BaseN frame. This means that the CNTVOFF address in any implemented CNTEL0BaseN frame is RAZ/WI.

In an implementation that supports 64-bit atomic accesses, a CNTVOFF{<n>} register must be accessible as an atomic 64-bit value.

CNTVOFF can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Range |
|-----------|----------|--------|-------|
| Timer | CNTBaseN | 0x018 | 31:0 |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Range |
|-----------|----------|--------|-------|
| Timer | CNTBaseN | 0x01C | 63:32 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CNTVOFF<n>, Counter-timer Virtual Offsets, n = 0 - 7

The CNTVOFF<n> characteristics are:

Purpose

Holds the 64-bit virtual offset for frame CNTBase<n>. This is the offset between real time and virtual time.

Configuration

The power domain of CNTVOFF<n> is IMPLEMENTATION DEFINED.

Implementation of this register is OPTIONAL.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

Attributes

CNTVOFF<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Virtual offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Virtual offset | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Virtual offset.

The reset behavior of this field is:

- On a Timer reset, this field resets to an architecturally UNKNOWN value.

Accessing CNTVOFF<n>

In the CNTCTLBase frame a CNTVOFF<n> register must be implemented, as a RW register, for each CNTBaseN frame that has virtual timer capability. For more information, see 'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames'.

Note

The value of <n> in an instance of CNTVOFF<n> specifies the value of N for the associated CNTBaseN frame.

In a system that recognizes two Security states, for any CNTVOFF<n> register in the CNTCTLBase frame:

- CNTVOFF<n> is always accessible by Secure accesses.
- [CNTNSAR.NS<n>](#) determines whether CNTVOFF<n> is accessible by Non-secure accesses.

The register location of any unimplemented CNTVOFF<n> register in the CNTCTLBase frame is RAZ/WI.

The CNTVOFF<n> register is accessible in the CNTBaseN frame using [CNTVOFF](#).

In an implementation that supports 64-bit atomic accesses, then the CNTVOFF<n> registers must be accessible as atomic 64-bit values.

CNTVOFF<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Range |
|-----------|------------|-------------------|-------|
| Timer | CNTCTLBase | $0x080 + (8 * n)$ | 31:0 |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Range |
|-----------|------------|-------------------|-------|
| Timer | CNTCTLBase | $0x084 + (8 * n)$ | 63:32 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CounterID<n>, Counter ID registers, n = 0 - 11

The CounterID<n> characteristics are:

Purpose

IMPLEMENTATION DEFINED identification registers 0 to 11 for the memory-mapped Generic Timer.

Configuration

The power domain of CounterID<n> is IMPLEMENTATION DEFINED.

For more information, see 'Power and reset domains for the system level implementation of the Generic Timer'.

These registers are implemented independently in each of the implemented Generic Timer memory-mapped frames.

If the implementation of the Counter ID registers requires an architecture version, the value for this version of the Arm Generic Timer is version 0.

The Counter ID registers can be implemented as a set of CoreSight ID registers, comprising Peripheral ID Registers and Component ID Registers. An implementation of these registers for the Generic Timer must use a Component class value of 0xF.

Attributes

CounterID<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing CounterID<n>

These registers must be implemented, as RO registers, in every implemented Generic Timer memory-mapped frame.

For the CNTCTLBase frame, in a system that recognizes two Security states these registers are accessible by both Secure and Non-secure accesses.

For the CNTControlBase frame, in a system that supports Secure and Non-secure memory maps the frame is implemented only in the Secure memory map, meaning these registers are implemented only in the Secure memory map.

For the CNTBaseN frames, 'CNTCTLBase status and control fields for the CNTBaseN and CNTEL0BaseN frames' describes the status fields that identify whether a frame is implemented, and for an implemented frame:

- Whether the CNTBaseN frame has virtual timer capability.
- Whether the corresponding CNTEL0BaseN frame is implemented.
- For an implementation that recognizes two Security states, whether the CNTBaseN frame, and any corresponding CNTEL0BaseN frame, is accessible by Non-secure accesses.

CounterID<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|-------|--------|----------|
|-----------|-------|--------|----------|

| | | | |
|-------|----------------|-----------------------|--------------|
| Timer | CNTControlBase | 0xFD0 + (4 * n) | CounterID<n> |
|-------|----------------|-----------------------|--------------|

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|-------------|--------------------|--------------|
| Timer | CNTReadBase | 0xFD0 + (4 * n) | CounterID<n> |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|----------|--------------------|--------------|
| Timer | CNTBaseN | 0xFD0 + (4 * n) | CounterID<n> |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|-------------|--------------------|--------------|
| Timer | CNTELOBaseN | 0xFD0 + (4 * n) | CounterID<n> |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|------------|--------------------|--------------|
| Timer | CNTCTLBase | 0xFD0 + (4 * n) | CounterID<n> |

Accesses on this interface are **RO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIAPPCLEAR, CTI Application Trigger Clear register

The CTIAPPCLEAR characteristics are:

Purpose

Clears bits of the Application Trigger register.

Configuration

CTIAPPCLEAR is in the Debug power domain.

Attributes

CTIAPPCLEAR is a 32-bit register.

Field descriptions

| | | | | | | | | |
|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 |
| APPCLEAR31 | APPCLEAR30 | APPCLEAR29 | APPCLEAR28 | APPCLEAR27 | APPCLEAR26 | APPCLEAR25 | APPCLEAR24 | APPCLEAR23 |

APPCLEAR<x>, bit [x], for x = 31 to 0

Application trigger <x> disable.

Bits [31:N] are RAZ/WI. N is the number of ECT channels implemented as defined by the [CTIDEVID](#).NUMCHAN field.

Writing to this bit has the following effect:

| APPCLEAR<x> | Meaning |
|-------------|---|
| 0b0 | No effect. |
| 0b1 | Clear corresponding bit in CTIAPPTRIG to 0 and clear the corresponding channel event. |

If the ECT does not support multicycle channel events, use of CTIAPPCLEAR is deprecated and the debugger must only use [CTIAPPULSE](#).

Accessing CTIAPPCLEAR

CTIAPPCLEAR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-------------|
| CTI | 0x018 | CTIAPPCLEAR |

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **WI**.
- When !SoftwareLockStatus() accesses to this register are **WO**.

CTIAPPPULSE, CTI Application Pulse register

The CTIAPPPULSE characteristics are:

Purpose

Causes event pulses to be generated on ECT channels.

Configuration

CTIAPPPULSE is in the Debug power domain.

Attributes

CTIAPPPULSE is a 32-bit register.

Field descriptions

| | | | | | | | | |
|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|----------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 |
| APPPULSE31 | APPPULSE30 | APPPULSE29 | APPPULSE28 | APPPULSE27 | APPPULSE26 | APPPULSE25 | APPPULSE24 | APPPULSE23 |

APPPULSE<x>, bit [x], for x = 31 to 0

Generate event pulse on ECT channel <x>.

Bits [31:N] are RAZ/WI. N is the number of ECT channels implemented as defined by the [CTIDEVID](#).NUMCHAN field.

Writing to this bit has the following effect:

| APPPULSE<x> | Meaning |
|-------------|------------------------------------|
| 0b0 | No effect. |
| 0b1 | Channel <x> event pulse generated. |

Note

- The CTIAPPPULSE operation does not affect the state of the Application Trigger register, CTIAPPTRIG. If the channel is active, either because of an earlier event or from the application trigger, then the value written to CTIAPPPULSE might have no effect.
- Multiple pulse events that occur close together might be merged into a single pulse event.

Accessing CTIAPPPULSE

It is CONSTRAINED UNPREDICTABLE whether a write to CTIAPPPULSE generates an event on a channel if CTICONTROL.GLBEN is 0.

CTIAPPPULSE can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-------------|
| CTI | 0x01C | CTIAPPPULSE |

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **WI**.
- When !SoftwareLockStatus() accesses to this register are **WO**.

CTIAPPSET, CTI Application Trigger Set register

The CTIAPPSET characteristics are:

Purpose

Sets bits of the Application Trigger register.

Configuration

CTIAPPSET is in the Debug power domain.

Attributes

CTIAPPSET is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| APPSET31 | APPSET30 | APPSET29 | APPSET28 | APPSET27 | APPSET26 | APPSET25 | APPSET24 | APPSET23 | APPSET22 | APPSET21 | APPSET20 | APPSET19 | APPSET18 | APPSET17 | APPSET16 | APPSET15 | APPSET14 | APPSET13 | APPSET12 | APPSET11 | APPSET10 | APPSET9 | APPSET8 | APPSET7 | APPSET6 | APPSET5 | APPSET4 | APPSET3 | APPSET2 | APPSET1 | APPSET0 |

APPSET<x>, bit [x], for x = 31 to 0

Application trigger <x> enable.

Bits [31:N] are RAZ/WI. N is the number of ECT channels implemented as defined by the [CTIDEVID](#).NUMCHAN field.

| APPSET<x> | Meaning |
|-----------|--|
| 0b0 | Reading this means the application trigger is inactive. Writing this has no effect. |
| 0b1 | Reading this means the application trigger is active. Writing this sets the corresponding bit in CTIAPPTRIG to 1 and generates a channel event. |

If the ECT does not support multicycle channel events, use of CTIAPPSET is deprecated and the debugger must only use [CTIAPPULSE](#).

The reset behavior of this field is:

- On an External debug reset, this field resets to an architecturally UNKNOWN value.

Accessing CTIAPPSET

CTIAPPSET can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| CTI | 0x014 | CTIAPPSET |

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **RO**.
- When !SoftwareLockStatus() accesses to this register are **RW**.

CTIAUTHSTATUS, CTI Authentication Status register

The CTIAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for CTI.

Configuration

CTIAUTHSTATUS is in the Debug power domain.

This register is OPTIONAL, and is required for CoreSight compliance.

Attributes

CTIAUTHSTATUS is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-----|----|----|----|------|----|----|----|-----|----|----|----|------|----|----|----|-----|----|---|---|-------|---|------|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | RAZ | | | | RES0 | | | | RAZ | | | | RES0 | | | | RAZ | | | | NSNID | | NSID | | | | | |

Bits [31:28]

Reserved, RES0.

Bits [27:24]

Reserved, RAZ.

Bits [23:16]

Reserved, RES0.

Bits [15:12]

Reserved, RAZ.

Bits [11:8]

Reserved, RES0.

Bits [7:4]

Reserved, RAZ.

NSNID, bits [3:2]

If EL3 is implemented, this field holds the same value as [DBGAUTHSTATUS_EL1](#).NSNID.

If EL3 is not implemented and the implemented Security state is Secure state, this field holds the same value as [DBGAUTHSTATUS_EL1](#).SNID.

NSID, bits [1:0]

If EL3 is implemented, this field holds the same value as [DBGAUTHSTATUS_EL1](#).NSID.

If EL3 is not implemented and the implemented Security state is Secure state, this field holds the same value as [DBGAUTHSTATUS_EL1](#).SID.

Accessing CTIAUTHSTATUS

CTIAUTHSTATUS can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|---------------|
| CTI | 0xFB8 | CTIAUTHSTATUS |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTICHINSTATUS, CTI Channel In Status register

The CTICHINSTATUS characteristics are:

Purpose

Provides the raw status of the ECT channel inputs to the CTI.

Configuration

CTICHINSTATUS is in the Debug power domain.

Attributes

CTICHINSTATUS is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 |
| CHIN31 | CHIN30 | CHIN29 | CHIN28 | CHIN27 | CHIN26 | CHIN25 | CHIN24 | CHIN23 | CHIN22 | CHIN21 | CHIN20 | CHIN19 | CHIN18 | CHIN17 |

CHIN<n>, bit [n], for n = 31 to 0

Input channel <n> status.

Bits [31:N] are RAZ. N is the number of ECT channels implemented as defined by the [CTIDEVID](#).NUMCHAN field.

| CHIN<n> | Meaning |
|---------|--------------------------------|
| 0b0 | Input channel <n> is inactive. |
| 0b1 | Input channel <n> is active. |

If the ECT channels do not support multicycle events then it is IMPLEMENTATION DEFINED whether an input channel can be observed as active.

Accessing CTICHINSTATUS

CTICHINSTATUS can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|---------------|
| CTI | 0x138 | CTICHINSTATUS |

Accesses on this interface are **RO**.

CTICHOUTSTATUS, CTI Channel Out Status register

The CTICHOUTSTATUS characteristics are:

Purpose

Provides the status of the ECT channel outputs from the CTI.

Configuration

CTICHOUTSTATUS is in the Debug power domain.

Attributes

CTICHOUTSTATUS is a 32-bit register.

Field descriptions

| | | | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| CHOUT31 | CHOUT30 | CHOUT29 | CHOUT28 | CHOUT27 | CHOUT26 | CHOUT25 | CHOUT24 | CHOUT23 | CHOUT22 | CHOUT21 | CHOUT20 |

CHOUT<n>, bit [n], for n = 31 to 0

Output channel <n> status.

Bits [31:N] are RAZ. N is the number of ECT channels implemented as defined by the [CTIDEVID](#).NUMCHAN field.

Possible values of this bit are:

| CHOUT<n> | Meaning |
|----------|---------------------------------|
| 0b0 | Output channel <n> is inactive. |
| 0b1 | Output channel <n> is active. |

If the ECT channels do not support multicycle events then it is IMPLEMENTATION DEFINED whether an output channel can be observed as active.

Note

The value in CTICHOUTSTATUS is after gating by the channel gate. For more information, see [CTIGATE](#).

Accessing CTICHOUTSTATUS

CTICHOUTSTATUS can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------------|
| CTI | 0x13C | CTICHOUTSTATUS |

Accesses on this interface are **RO**.

CTICIDR0, CTI Component Identification Register 0

The CTICIDR0 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Component Identification scheme'.

Configuration

CTICIDR0 is in the Debug power domain.

Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTICIDR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL 0 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Preamble.

Reads as 0x0D.

Access to this field is **RO**.

Accessing CTICIDR0

CTICIDR0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| CTI | 0xFF0 | CTICIDR0 |

Accesses on this interface are **RO**.

CTICIDR1, CTI Component Identification Register 1

The CTICIDR1 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Component Identification scheme'.

Configuration

CTICIDR1 is in the Debug power domain.

Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTICIDR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---|---------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | CLASS | | | | PRMBL_1 | | | |

Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

| CLASS | Meaning |
|--------|----------------------|
| 0b1001 | CoreSight component. |

Other values are defined by the CoreSight Architecture.

This field reads as 0x9.

PRMBL_1, bits [3:0]

Preamble. RAZ.

Reads as 0b0000.

Access to this field is **RO**.

Accessing CTICIDR1

CTICIDR1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| CTI | 0xFF4 | CTICIDR1 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTICIDR2, CTI Component Identification Register 2

The CTICIDR2 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Component Identification scheme'.

Configuration

CTICIDR2 is in the Debug power domain.

Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTICIDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL 2 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Preamble.

Reads as 0x05.

Access to this field is **RO**.

Accessing CTICIDR2

CTICIDR2 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| CTI | 0xFF8 | CTICIDR2 |

Accesses on this interface are **RO**.

CTICIDR3, CTI Component Identification Register 3

The CTICIDR3 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Component Identification scheme'.

Configuration

CTICIDR3 is in the Debug power domain.

Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTICIDR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL 3 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Preamble.

Reads as 0xB1.

Access to this field is **RO**.

Accessing CTICIDR3

CTICIDR3 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| CTI | 0xFFC | CTICIDR3 |

Accesses on this interface are **RO**.

CTICLAIMCLR, CTI CLAIM Tag Clear register

The CTICLAIMCLR characteristics are:

Purpose

Used by software to read the values of the CLAIM bits, and to clear CLAIM tag bits to 0.

Configuration

CTICLAIMCLR is in the Debug power domain.

Implementation of this register is OPTIONAL.

Attributes

CTICLAIMCLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 |
| CLAIM31 | CLAIM30 | CLAIM29 | CLAIM28 | CLAIM27 | CLAIM26 | CLAIM25 | CLAIM24 | CLAIM23 | CLAIM22 | CLAIM21 | CLAIM20 | CLAIM19 |

CLAIM<x>, bit [x], for x = 31 to 0

CLAIM tag clear bit.

Reads return the value of CLAIM<x>, writes have the following behavior:

| CLAIM<x> | Meaning |
|----------|---------------------------------|
| 0b0 | No action. |
| 0b1 | Indirectly clear CLAIM<x> to 0. |

A single write to CTICLAIMCLR can clear multiple tags to 0.

If x is greater than or equal to the IMPLEMENTATION DEFINED number of CLAIM tags, this bit is RAZ/WI.

An External Debug reset clears the CLAIM tag bits to 0.

Accessing CTICLAIMCLR

CTICLAIMCLR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-------------|
| CTI | 0xFA4 | CTICLAIMCLR |

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **RO**.
- When !SoftwareLockStatus() accesses to this register are **RW**.

CTICLAIMSET, CTI CLAIM Tag Set register

The CTICLAIMSET characteristics are:

Purpose

Used by software to set CLAIM bits to 1.

Configuration

CTICLAIMSET is in the Debug power domain.

Implementation of this register is OPTIONAL.

Attributes

CTICLAIMSET is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 |
| CLAIM31 | CLAIM30 | CLAIM29 | CLAIM28 | CLAIM27 | CLAIM26 | CLAIM25 | CLAIM24 | CLAIM23 | CLAIM22 | CLAIM21 | CLAIM20 | CLAIM19 |

CLAIM<x>, bit [x], for x = 31 to 0

CLAIM tag set bit.

If x is less than the IMPLEMENTATION DEFINED number of CLAIM tags, this field is RAO and the behavior on writes is:

| CLAIM<x> | Meaning |
|----------|-----------------------------------|
| 0b0 | No action. |
| 0b1 | Indirectly set CLAIM<x> tag to 1. |

A single write to CTICLAIMSET can set multiple tags to 1.

If x is greater than or equal to the IMPLEMENTATION DEFINED number of CLAIM tags, this bit is RAZ/WI.

An External Debug reset clears the CLAIM tag bits to 0.

Accessing CTICLAIMSET

CTICLAIMSET can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-------------|
| CTI | 0xFA0 | CTICLAIMSET |

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **RO**.
- When !SoftwareLockStatus() accesses to this register are **RW**.

CTICONTROL, CTI Control register

The CTICONTROL characteristics are:

Purpose

Controls whether the CTI is enabled.

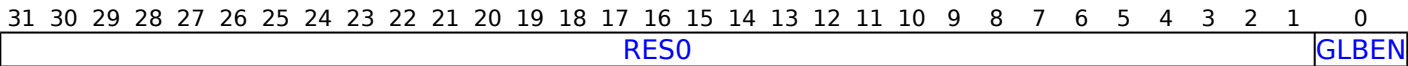
Configuration

CTICONTROL is in the Debug power domain.

Attributes

CTICONTROL is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

GLBEN, bit [0]

Enables or disables the CTI mapping functions. Possible values of this field are:

| GLBEN | Meaning |
|-------|---|
| 0b0 | CTI mapping functions and application trigger disabled. |
| 0b1 | CTI mapping functions and application trigger enabled. |

When GLBEN is 0, the input channel to output trigger, input trigger to output channel, and application trigger functions are disabled and do not signal new events on either output triggers or output channels. If a previously asserted output trigger has not been acknowledged, it is CONSTRAINED UNPREDICTABLE which of the following occurs:

- The output trigger remains asserted after the mapping functions are disabled.
- The output trigger is deasserted after the mapping functions are disabled.

All output triggers are disabled by CTI reset.

If the ECT supports multicycle channel events any existing output channel events will be terminated.

The reset behavior of this field is:

- On an External debug reset, this field resets to 0.

Accessing CTICONTROL

CTICONTROL can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|------------|
| CTI | 0x000 | CTICONTROL |

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **RO**.
- When !SoftwareLockStatus() accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIDEVAFF0, CTI Device Affinity register 0

The CTIDEVAFF0 characteristics are:

Purpose

Copy of the low half of the PE [MPIDR_EL1](#) register that allows a debugger to determine which PE in a multiprocessor system the CTI component relates to.

Configuration

CTIDEVAFF0 is in the Debug power domain.

If the CTI is CTIv1, this register is OPTIONAL. If the CTI is CTIv2, this register is mandatory.

Arm recommends that the CTI is CTIv2.

In an Armv8.5 compliant implementation, the CTI must be CTIv2.

If this register is implemented, then [CTIDEVAFF1](#) must also be implemented. If the CTI of a PE does not implement the CTI Device Affinity registers, the CTI block of the external debug memory map must be located 64KB above the debug registers in the external debug interface.

Attributes

CTIDEVAFF0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| MPIDR_EL1lo | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

MPIDR_EL1lo, bits [31:0]

[MPIDR_EL1](#) low half. Read-only copy of the low half of [MPIDR_EL1](#), as seen from the highest implemented Exception level.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIDEVAFF0

CTIDEVAFF0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|------------|
| CTI | 0xFA8 | CTIDEVAFF0 |

Accesses on this interface are **RO**.

CTIDEVAFF1, CTI Device Affinity register 1

The CTIDEVAFF1 characteristics are:

Purpose

Copy of the high half of the PE [MPIDR_EL1](#) register that allows a debugger to determine which PE in a multiprocessor system the CTI component relates to.

Configuration

CTIDEVAFF1 is in the Debug power domain.

If the CTI is CTIv1, this register is OPTIONAL. If the CTI is CTIv2, this register is mandatory.

Arm recommends that the CTI is CTIv2.

In an Armv8.5 compliant implementation, the CTI must be CTIv2.

If this register is implemented, then [CTIDEVAFF0](#) must also be implemented. If the CTI of a PE does not implement the CTI Device Affinity registers, the CTI block of the external debug memory map must be located 64KB above the debug registers in the external debug interface.

Attributes

CTIDEVAFF1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------------------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | MPIDR_EL1hi | | | | | | | | | | | | | | | |

MPIDR_EL1hi, bits [31:0]

[MPIDR_EL1](#) high half. Read-only copy of the high half of [MPIDR_EL1](#), as seen from the highest implemented Exception level.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIDEVAFF1

CTIDEVAFF1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|------------|
| CTI | 0xFAC | CTIDEVAFF1 |

Accesses on this interface are **RO**.

CTIDEVARCH, CTI Device Architecture register

The CTIDEVARCH characteristics are:

Purpose

Identifies the programmers' model architecture of the CTI component.

Configuration

CTIDEVARCH is in the Debug power domain.

If the CTI is CTIv1, this register is OPTIONAL. If the CTI is CTIv2, this register is mandatory.

Arm recommends that the CTI is CTIv2.

In an Armv8.5 compliant implementation, the CTI must be CTIv2.

If this register is not implemented, [CTIDEVAFF0](#) and [CTIDEVAFF1](#) are also not implemented.

Attributes

CTIDEVARCH is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|---------|----------|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ARCHITECT | | | | | | | | | | | PRESENT | REVISION | | | | ARCHID | | | | | | | | | | | | | | | |

ARCHITECT, bits [31:21]

Defines the architecture of the component. For CTI, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, 0x4.

Bits [27:21] are the JEP106 ID code, 0x3B.

Reads as 0b01000111011.

Access to this field is **RO**.

PRESENT, bit [20]

Indicates that the DEVARCH is present.

Reads as 0b1.

Access to this field is **RO**.

REVISION, bits [19:16]

When FEAT_DoPD is implemented:

Revision.

Defines the architecture revision of the component.

| REVISION | Meaning |
|----------|--|
| 0b0000 | First revision. |
| 0b0001 | As 0b0000, and also adds support for CTIDEVCTL . |

All other values are reserved.

Access to this field is **RO**.

Otherwise:

Revision.

Defines the architecture revision of the component.

All other values are reserved.

Reads as 0b0000.

Access to this field is **RO**.

ARCHID, bits [15:0]

Defines this part to be an Armv8 debug component. For architectures defined by Arm this is further subdivided.

For CTI:

- Bits [15:12] are the architecture version, 0x1.
- Bits [11:0] are the architecture part number, 0xA14.

This corresponds to CTI architecture version CTIv2.

Reads as 0x1A14.

Access to this field is **RO**.

Accessing CTIDEVARCH

CTIDEVARCH can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|------------|
| CTI | 0xFBC | CTIDEVARCH |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIDEVCTL, CTI Device Control register

The CTIDEVCTL characteristics are:

Purpose

Provides target-specific device controls

Configuration

CTIDEVCTL is in the Debug power domain.

This register is present only when FEAT_DoPD is implemented. Otherwise, direct accesses to CTIDEVCTL are RES0.

Attributes

CTIDEVCTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|-------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | RCE | | OSUCE | | | | | | | | | | | | | |

Bits [31:2]

Reserved, RES0.

RCE, bit [1]

Reset Catch Enable.

| RCE | Meaning |
|-----|-----------------------------------|
| 0b0 | Reset Catch debug event disabled. |
| 0b1 | Reset Catch debug event enabled. |

The reset behavior of this field is:

- On an External debug reset, this field resets to 0.

OSUCE, bit [0]

OS Unlock Catch Enable

| OSUCE | Meaning |
|-------|---------------------------------------|
| 0b0 | OS Unlock Catch debug event disabled. |
| 0b1 | OS Unlock Catch debug event enabled. |

The reset behavior of this field is:

- On an External debug reset, this field resets to 0.

Accessing CTIDEVCTL

CTIDEVCTL can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
|-----------|--------|----------|

| | | |
|-----|-------|-----------|
| CTI | 0x150 | CTIDEVCTL |
|-----|-------|-----------|

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **RO**.
- When !SoftwareLockStatus() accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIDEVID, CTI Device ID register 0

The CTIDEVID characteristics are:

Purpose

Describes the CTI component to the debugger.

Configuration

CTIDEVID is in the Debug power domain.

Attributes

CTIDEVID is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|-------|------|----|----|----|---------|----|----|----|------|---------|----|----|----|------|-----------|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | INOUT | RES0 | | | | NUMCHAN | | | | RES0 | NUMTRIG | | | | RES0 | EXTMUXNUM | | | | | | | | | | |

Bits [31:26]

Reserved, RES0.

INOUT, bits [25:24]

Input/output options. Indicates presence of the input gate. If the CTM is not implemented or CTIv2 is not implemented, this field is RAZ.

| INOUT | Meaning |
|-------|---|
| 0b00 | CTIGATE does not mask propagation of input events from external channels. |
| 0b01 | CTIGATE masks propagation of input events from external channels. |

All other values are reserved.

Bits [23:22]

Reserved, RES0.

NUMCHAN, bits [21:16]

Number of ECT channels implemented. For Armv8, valid values are:

- 0b000011 3 channels (0..2) implemented.
- 0b000100 4 channels (0..3) implemented.
- 0b000101 5 channels (0..4) implemented.
- 0b000110 6 channels (0..5) implemented.

and so on up to 0b100000, 32 channels (0..31) implemented.

All other values are reserved.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [15:14]

Reserved, RES0.

NUMTRIG, bits [13:8]

Number of triggers implemented. This is one more than the index of the largest trigger, rather than the actual number of triggers. For Armv8, valid values are:

- 0b000011 Up to 3 triggers (0..2) implemented.
- 0b001000 Up to 8 triggers (0..7) implemented.
- 0b001001 Up to 9 triggers (0..8) implemented.
- 0b001010 Up to 10 triggers (0..9) implemented.

and so on up to 0b100000, 32 triggers (0..31) implemented.

All other values are reserved. If the PE contains a Trace extension, this field must be at least 0b001000. There is no guarantee that any of the implemented triggers, including the highest numbered, are connected to any components.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [7:5]

Reserved, RES0.

EXTMUXNUM, bits [4:0]

Number of multiplexors available on triggers. This value is used in conjunction with External Control register, [ASICCTL](#).

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIDEVID

CTIDEVID can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| CTI | 0xFC8 | CTIDEVID |

Accesses on this interface are **RO**.

CTIDEVID1, CTI Device ID register 1

The CTIDEVID1 characteristics are:

Purpose

Reserved for future information about the CTI component to the debugger.

Configuration

CTIDEVID1 is in the Debug power domain.

Attributes

CTIDEVID1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |

Bits [31:0]

Reserved, RES0.

Accessing CTIDEVID1

CTIDEVID1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| CTI | 0xFC4 | CTIDEVID1 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIDEVID2, CTI Device ID register 2

The CTIDEVID2 characteristics are:

Purpose

Reserved for future information about the CTI component to the debugger.

Configuration

CTIDEVID2 is in the Debug power domain.

Attributes

CTIDEVID2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |

Bits [31:0]

Reserved, RES0.

Accessing CTIDEVID2

CTIDEVID2 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| CTI | 0xFC0 | CTIDEVID2 |

Accesses on this interface are **RO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIDEVTYPE, CTI Device Type register

The CTIDEVTYPE characteristics are:

Purpose

Indicates to a debugger that this component is part of a PEs cross-trigger interface.

Configuration

CTIDEVTYPE is in the Debug power domain.

Implementation of this register is OPTIONAL.

Attributes

CTIDEVTYPE is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|---|---|---|-----|---|---|-------|---|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | SUB | | | MAJOR | | |

Bits [31:8]

Reserved, RES0.

SUB, bits [7:4]

Subtype. Indicates this is a component within a PE.

Reads as 0b0001.

Access to this field is **RO**.

MAJOR, bits [3:0]

Major type. Indicates this is a cross-trigger component.

Reads as 0b0100.

Access to this field is **RO**.

Accessing CTIDEVTYPE

CTIDEVTYPE can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|------------|
| CTI | 0xFCC | CTIDEVTYPE |

Accesses on this interface are **RO**.

CTIGATE, CTI Channel Gate Enable register

The CTIGATE characteristics are:

Purpose

Determines whether events on channels propagate through the CTM to other ECT components, or from the CTM into the CTI.

Configuration

CTIGATE is in the Debug power domain.

Attributes

CTIGATE is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| GATE31 | GATE30 | GATE29 | GATE28 | GATE27 | GATE26 | GATE25 | GATE24 | GATE23 | GATE22 | GATE21 | GATE20 | GATE19 | GATE18 | GATE17 | GATE16 | GATE15 | GATE14 | GATE13 | GATE12 | GATE11 | GATE10 | GATE9 | GATE8 | GATE7 | GATE6 | GATE5 | GATE4 | GATE3 | GATE2 | GATE1 | GATE0 |

GATE<x>, bit [x], for x = 31 to 0

Channel <x> gate enable.

Bits [31:N] are RAZ/WI. N is the number of ECT channels implemented as defined by the [CTIDEVID](#).NUMCHAN field.

| GATE<x> | Meaning |
|---------|--|
| 0b0 | Disable output and, if CTIDEVID .INOUT == 0b01, input channel <x> propagation. |
| 0b1 | Enable output and, if CTIDEVID .INOUT == 0b01, input channel <x> propagation. |

If GATE<x> is set to 0, no new events will be propagated to the ECT, and if the ECT supports multicycle channel events any existing output channel events will be terminated.

The reset behavior of this field is:

- On an External debug reset, this field resets to an architecturally UNKNOWN value.

Accessing CTIGATE

CTIGATE can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| CTI | 0x140 | CTIGATE |

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **RO**.
- When !SoftwareLockStatus() accesses to this register are **RW**.

CTIINEN<n>, CTI Input Trigger to Output Channel Enable registers, n = 0 - 31

The CTIINEN<n> characteristics are:

Purpose

Enables the signaling of an event on output channels when input trigger event n is received by the CTI.

Configuration

CTIINEN<n> is in the Debug power domain.

If input trigger n is not connected, the behavior of CTIINEN<n> is IMPLEMENTATION DEFINED.

Attributes

CTIINEN<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 |
| INEN31 | INEN30 | INEN29 | INEN28 | INEN27 | INEN26 | INEN25 | INEN24 | INEN23 | INEN22 | INEN21 | INEN20 | INEN19 | INEN18 | INEN17 |

INEN<x>, bit [x], for x = 31 to 0

Input trigger <n> to output channel <x> enable.

Bits [31:N] are RAZ/WI. N is the number of ECT channels implemented as defined by the [CTIDEVID](#).NUMCHAN field.

| INEN<x> | Meaning |
|---------|---|
| 0b0 | Input trigger <n> will not generate an event on output channel <x>. |
| 0b1 | Input trigger <n> will generate an event on output channel <x>. |

The reset behavior of this field is:

- On an External debug reset, this field resets to an architecturally UNKNOWN value.

Accessing CTIINEN<n>

CTIINEN<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|------------|
| CTI | 0x020 + (4 * n) | CTIINEN<n> |

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **RO**.
- When !SoftwareLockStatus() accesses to this register are **RW**.

CTIINTACK, CTI Output Trigger Acknowledge register

The CTIINTACK characteristics are:

Purpose

Can be used to deactivate the output triggers.

Configuration

CTIINTACK is in the Debug power domain.

Attributes

CTIINTACK is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 |
| ACK31 | ACK30 | ACK29 | ACK28 | ACK27 | ACK26 | ACK25 | ACK24 | ACK23 | ACK22 | ACK21 | ACK20 | ACK19 | ACK18 | ACK17 | ACK16 | ACK15 |

ACK<n>, bit [n], for n = 31 to 0

Acknowledge for output trigger <n>.

Bits [31:N] are RAZ/WI. N is the number of CTI triggers implemented as defined by the [CTIDEVID](#).NUMTRIG field.

If any of the following is true, writes to ACK<n> are ignored:

- $n \geq$ [CTIDEVID](#).NUMTRIG, the number of implemented triggers.
- Output trigger n is not active.
- The channel mapping function output, as controlled by [CTIOUTEN<n>](#), is still active.

Otherwise, if any of the following are true, it is IMPLEMENTATION DEFINED whether writes to ACK<n> are ignored:

- Output trigger n is not implemented.
- Output trigger n is not connected.
- Output trigger n is self-acknowledging and does not require software acknowledge.

Otherwise, the behavior on writes to ACK<n> is as follows:

| ACK<n> | Meaning |
|--------|-------------------------|
| 0b0 | No effect |
| 0b1 | Deactivate the trigger. |

Accessing CTIINTACK

A debugger must read [CTITRIGOUTSTATUS](#) to confirm that the output trigger has been acknowledged before generating any event that must be ordered after the write to CTIINTACK, such as a write to CTIAPPPULSE to activate another trigger.

CTIINTACK can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| CTI | 0x010 | CTIINTACK |

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **WI**.
- When !SoftwareLockStatus() accesses to this register are **WO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIITCTRL, CTI Integration mode Control register

The CTIITCTRL characteristics are:

Purpose

Enables the CTI to switch from its default mode into integration mode, where test software can control directly the inputs and outputs of the PE, for integration testing or topology detection.

Configuration

It is IMPLEMENTATION DEFINED whether CTIITCTRL is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is OPTIONAL.

Attributes

CTIITCTRL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | IME | | | | | | | | | | | | | | | |

Bits [31:1]

Reserved, RES0.

IME, bit [0]

Integration mode enable. When IME == 1, the device reverts to an integration mode to enable integration testing or topology detection. The integration mode behavior is IMPLEMENTATION DEFINED.

| IME | Meaning |
|-----|---------------------------|
| 0b0 | Normal operation. |
| 0b1 | Integration mode enabled. |

The following resets apply:

- If the register is implemented in the Core power domain:
 - On a Cold reset, this field resets to 0.
 - On an External debug reset, the value of this field is unchanged.
 - On a Warm reset, the value of this field is unchanged.
- If the register is implemented in the External debug power domain:
 - On a Cold reset, the value of this field is unchanged.
 - On an External debug reset, this field resets to 0.
 - On a Warm reset, the value of this field is unchanged.

Accessing CTIITCTRL

CTIITCTRL can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| CTI | 0xF00 | CTIITCTRL |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register are **IMPDEF**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTILAR, CTI Lock Access Register

The CTILAR characteristics are:

Purpose

Allows or disallows access to the CTI registers through a memory-mapped interface.

The optional Software Lock provides a lock to prevent memory-mapped writes to the Cross-Trigger Interface registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the Cross-Trigger Interface registers. It does not, and cannot, prevent all accidental or malicious damage.

Configuration

CTILAR is in the Debug power domain.

If FEAT_Debugv8p4 is implemented, the Software Lock is not implemented.

Software uses CTILAR to set or clear the lock, and [CTILSR](#) to check the current status of the lock.

Attributes

CTILAR is a 32-bit register.

Field descriptions

When Software Lock is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | KEY | | | | | | | | | | | | | | | |

KEY, bits [31:0]

Lock Access control. Writing the key value 0xC5ACCE55 to this field unlocks the lock, enabling write accesses to this component's registers through a memory-mapped interface.

Writing any other value to this register locks the lock, disabling write accesses to this component's registers through a memory mapped interface.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |

Otherwise

Bits [31:0]

Reserved, RES0.

Accessing CTILAR

CTILAR can be accessed through a memory-mapped access to the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
|-----------|--------|----------|

CTILAR, CTI Lock Access Register

| | | |
|-----|-------|--------|
| CTI | 0xFB0 | CTILAR |
|-----|-------|--------|

Accesses on this interface are **WO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTILSR, CTI Lock Status Register

The CTILSR characteristics are:

Purpose

Indicates the current status of the Software Lock for CTI registers.

The optional Software Lock provides a lock to prevent memory-mapped writes to the Cross-Trigger Interface registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the Cross-Trigger Interface registers. It does not, and cannot, prevent all accidental or malicious damage.

Configuration

CTILSR is in the Debug power domain.

If FEAT_Debugv8p4 is implemented, the Software Lock is not implemented.

Software uses [CTILAR](#) to set or clear the lock, and CTILSR to check the current status of the lock.

Attributes

CTILSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|-----|-----|-----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | nTT | SLK | SLI |

Bits [31:3]

Reserved, RES0.

nTT, bit [2]

Not thirty-two bit access required. RAZ.

SLK, bit [1]

When Software Lock is implemented:

Software Lock status for this component. For an access to LSR that is not a memory-mapped access, or when the Software Lock is not implemented, this field is RES0.

For memory-mapped accesses when the Software Lock is implemented, possible values of this field are:

| SLK | Meaning |
|-----|---|
| 0b0 | Lock clear. Writes are permitted to this component's registers. |
| 0b1 | Lock set. Writes to this component's registers are ignored, and reads have no side effects. |

The reset behavior of this field is:

- On an External debug reset, this field resets to 1.

Otherwise:

Reserved, RAZ.

SLI, bit [0]

Software Lock implemented. For an access to LSR that is not a memory-mapped access, this field is RAZ. For memory-mapped accesses, the value of this field is IMPLEMENTATION DEFINED. Permitted values are:

| SLI | Meaning |
|-----|--|
| 0b0 | Software Lock not implemented or not memory-mapped access. |
| 0b1 | Software Lock implemented and memory-mapped access. |

Accessing CTILSR

CTILSR can be accessed through a memory-mapped access to the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| CTI | 0xFB4 | CTILSR |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIOUTEN<n>, CTI Input Channel to Output Trigger Enable registers, n = 0 - 31

The CTIOUTEN<n> characteristics are:

Purpose

Defines which input channels generate output trigger n.

Configuration

CTIOUTEN<n> is in the Debug power domain.

If output trigger n is not connected, the behavior of CTIOUTEN<n> is IMPLEMENTATION DEFINED.

Attributes

CTIOUTEN<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| OUTEN31 | OUTEN30 | OUTEN29 | OUTEN28 | OUTEN27 | OUTEN26 | OUTEN25 | OUTEN24 | OUTEN23 | OUTEN22 | OUTEN21 | OUTEN20 |

OUTEN<x>, bit [x], for x = 31 to 0

Input channel <x> to output trigger <n> enable.

Bits [31:N] are RAZ/WI. N is the number of ECT channels implemented as defined by the [CTIDEVID](#).NUMCHAN field.

Possible values of this bit are:

| OUTEN<x> | Meaning |
|----------|---|
| 0b0 | An event on input channel <x> will not cause output trigger <n> to be asserted. |
| 0b1 | An event on input channel <x> will cause output trigger <n> to be asserted. |

The reset behavior of this field is:

- On an External debug reset, this field resets to an architecturally UNKNOWN value.

Accessing CTIOUTEN<n>

CTIOUTEN<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|-------------|
| CTI | 0x0A0 + (4 * n) | CTIOUTEN<n> |

This interface is accessible as follows:

- When SoftwareLockStatus() accesses to this register are **RO**.
- When !SoftwareLockStatus() accesses to this register are **RW**.

CTIPIDR0, CTI Peripheral Identification Register 0

The CTIPIDR0 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

CTIPIDR0 is in the Debug power domain.

Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTIPIDR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|--------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PART 0 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, least significant byte.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIPIDR0

CTIPIDR0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| CTI | 0xFE0 | CTIPIDR0 |

Accesses on this interface are **RO**.

CTIPIDR1, CTI Peripheral Identification Register 1

The CTIPIDR1 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

CTIPIDR1 is in the Debug power domain.

Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTIPIDR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---|--------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | DES_0 | | | | PART_1 | | | |

Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, least significant nibble of JEP106 ID code. For Arm Limited, this field is 0b1011.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PART_1, bits [3:0]

Part number, most significant nibble.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIPIDR1

CTIPIDR1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| CTI | 0xFE4 | CTIPIDR1 |

Accesses on this interface are **RO**.

CTIPIDR2, CTI Peripheral Identification Register 2

The CTIPIDR2 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

CTIPIDR2 is in the Debug power domain.

Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTIPIDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|-------|----|-------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | REVISION | | | | JEDEC | | DES_1 | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Part major revision. Parts can also use this field to extend Part number to 16-bits.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

Indicates a JEP106 identity code is used.

Reads as 0b1.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, most significant bits of JEP106 ID code. For Arm Limited, this field is 0b011.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIPIDR2

CTIPIDR2 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| CTI | 0xFE8 | CTIPIDR2 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

CTIPIDR3, CTI Peripheral Identification Register 3

The CTIPIDR3 characteristics are:

Purpose

Provides information to identify a CTI component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

CTIPIDR3 is in the Debug power domain.

Implementation of this register is OPTIONAL.

This register is required for CoreSight compliance.

Attributes

CTIPIDR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | REVAND | | | | CMOD | | | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Part minor revision. Parts using [CTIPIDR2](#).REVISION as an extension to the Part number must use this field as a major revision number.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer modified. Indicates someone other than the Designer has modified the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing CTIPIDR3

CTIPIDR3 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| CTI | 0xFEC | CTIPIDR3 |

Accesses on this interface are **RO**.

CTIPIDR4, CTI Peripheral Identification Register 4

The CTIPIDR4 characteristics are:

Purpose

Provides information to identify a CTI component.
For more information, see 'About the Peripheral identification scheme'.

Configuration

CTIPIDR4 is in the Debug power domain.
Implementation of this register is OPTIONAL.
This register is required for CoreSight compliance.

Attributes

CTIPIDR4 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|------|---|---|---|-------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | SIZE | | | | DES_2 | | | |

Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

Size of the component. Log2 of the number of 4KB pages from the start of the component to the end of the component ID registers.
Reads as 0b0000.
Access to this field is **RO**.

DES_2, bits [3:0]

Designer, JEP106 continuation code, least significant nibble. For Arm Limited, this field is 0b0100.
This field has an IMPLEMENTATION DEFINED value.
Access to this field is **RO**.

Accessing CTIPIDR4

CTIPIDR4 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| CTI | 0xFD0 | CTIPIDR4 |

Accesses on this interface are **RO**.

CTITRIGINSTATUS, CTI Trigger In Status register

The CTITRIGINSTATUS characteristics are:

Purpose

Provides the status of the trigger inputs.

Configuration

CTITRIGINSTATUS is in the Debug power domain.

Attributes

CTITRIGINSTATUS is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | |
|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 |
| TRIN31 | TRIN30 | TRIN29 | TRIN28 | TRIN27 | TRIN26 | TRIN25 | TRIN24 | TRIN23 | TRIN22 | TRIN21 | TRIN20 | TRIN19 | TRIN18 | TRIN17 | TRIN16 |

TRIN<n>, bit [n], for n = 31 to 0

Trigger input <n> status.

Bits [31:N] are RAZ. N is the number of CTI triggers implemented as defined by the [CTIDEVID](#).NUMTRIG field.

| TRIN<n> | Meaning |
|---------|------------------------------|
| 0b0 | Input trigger n is inactive. |
| 0b1 | Input trigger n is active. |

Not implemented and not-connected input triggers are always inactive.

It is IMPLEMENTATION DEFINED whether an input trigger that does not support multicycle events can be observed as active.

Accessing CTITRIGINSTATUS

CTITRIGINSTATUS can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------------|
| CTI | 0x130 | CTITRIGINSTATUS |

Accesses on this interface are **RO**.

CTITRIGOUTSTATUS, CTI Trigger Out Status register

The CTITRIGOUTSTATUS characteristics are:

Purpose

Provides the raw status of the trigger outputs, after processing by any IMPLEMENTATION DEFINED trigger interface logic. For output triggers that are self-acknowledging, this is only meaningful if the CTI implements multicycle channel events.

Configuration

CTITRIGOUTSTATUS is in the Debug power domain.

Attributes

CTITRIGOUTSTATUS is a 32-bit register.

Field descriptions

| | | | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| TROUT31 | TROUT30 | TROUT29 | TROUT28 | TROUT27 | TROUT26 | TROUT25 | TROUT24 | TROUT23 | TROUT22 | TROUT21 | TROUT20 |

TROUT<n>, bit [n], for n = 31 to 0

Trigger output <n> status.

Bits [31:N] are RAZ. N is the value in [CTIDEVID.NUMTRIG](#).

If $n < N$, and output trigger <n> is implemented and connected, and either the trigger is not self-acknowledging or the CTI implements multicycle channel events, then permitted values for TROUT<n> are:

| TROUT<n> | Meaning |
|----------|-------------------------------|
| 0b0 | Output trigger n is inactive. |
| 0b1 | Output trigger n is active. |

Otherwise when $n < N$ it is IMPLEMENTATION DEFINED whether TROUT<n> behaves as described here or is RAZ.

Accessing CTITRIGOUTSTATUS

CTITRIGOUTSTATUS can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|------------------|
| CTI | 0x134 | CTITRIGOUTSTATUS |

Accesses on this interface are **RO**.

DBGAUTHSTATUS_EL1, Debug Authentication Status register

The DBGAUTHSTATUS_EL1 characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

Configuration

External register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to AArch64 System register [DBGAUTHSTATUS_EL1\[31:0\]](#).

External register DBGAUTHSTATUS_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGAUTHSTATUS\[31:0\]](#).

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

DBGAUTHSTATUS_EL1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|------|----|------|----|----|----|-------|----|----|----|------|----|------|----|----|----|------|---|-----|---|-------|---|------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | RTNID | | RTID | | RES0 | | | | RLNID | | | | RLID | | RES0 | | | | SNID | | SID | | NSNID | | NSID | | | |

Bits [31:28]

Reserved, RES0.

RTNID, bits [27:26]

Root non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RTID.

RTID, bits [25:24]

Root invasive debug.

| RTID | Meaning |
|------|---|
| 0b00 | Not implemented. |
| 0b10 | Implemented and disabled. ExternalRootInvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalRootInvasiveDebugEnabled() == TRUE. |

All other values are reserved.

If FEAT_RME is not implemented, the only permitted value is 00.

Bits [23:16]

Reserved, RES0.

RLNID, bits [15:14]

Realm non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RLID.

RLID, bits [13:12]

Realm invasive debug.

| RLID | Meaning |
|------|--|
| 0b00 | Not implemented. |
| 0b10 | Implemented and disabled. ExternalRealmInvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalRealmInvasiveDebugEnabled() == TRUE. |

All other values are reserved.

If FEAT_RME is not implemented, the only permitted value is 00.

Bits [11:8]

Reserved, RES0.

SNID, bits [7:6]

When FEAT_Debugv8p4 is implemented:

Secure non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.SID.

Otherwise:

Secure non-invasive debug.

| SNID | Meaning |
|------|---|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 1. |
| 0b10 | Implemented and disabled. ExternalSecureNoninvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalSecureNoninvasiveDebugEnabled() == TRUE. |

All other values are reserved.

SID, bits [5:4]

Secure invasive debug.

| SID | Meaning |
|------|---|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 1. |
| 0b10 | Implemented and disabled. ExternalSecureInvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalSecureInvasiveDebugEnabled() == TRUE. |

All other values are reserved.

NSNID, bits [3:2]**When FEAT_Debugv8p4 is implemented:**

Non-secure non-invasive debug.

| NSNID | Meaning |
|-------|---|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 0. |
| 0b11 | Implemented and enabled. ExternalNoninvasiveDebugEnabled() == TRUE. |

If the Effective value of [SCR_EL3.NS](#) is 1, or if EL3 is implemented and EL2 is not implemented, this field reads as 0b11.

All other values are reserved.

Otherwise:

Non-secure non-invasive debug.

| NSNID | Meaning |
|-------|---|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 0. |
| 0b10 | Implemented and disabled. ExternalNoninvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalNoninvasiveDebugEnabled() == TRUE. |

All other values are reserved.

NSID, bits [1:0]

Non-secure invasive debug.

| NSID | Meaning |
|------|---|
| 0b00 | Not implemented. EL3 is not implemented and the Effective value of SCR_EL3.NS is 0. |
| 0b10 | Implemented and disabled. ExternalInvasiveDebugEnabled() == FALSE. |
| 0b11 | Implemented and enabled. ExternalInvasiveDebugEnabled() == TRUE. |

All other values are reserved.

Accessing DBGAUTHSTATUS_EL1**DBGAUTHSTATUS_EL1 can be accessed through the external debug interface:**

| Component | Offset | Instance |
|-----------|--------|-------------------|
| Debug | 0xFB8 | DBGAUTHSTATUS_EL1 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

DBGBCR<n>_EL1, Debug Breakpoint Control Registers, n = 0 - 15

The DBGBCR<n>_EL1 characteristics are:

Purpose

Holds control information for a breakpoint. Forms breakpoint n together with value register [DBGBVR<n>_EL1](#).

Configuration

External register DBGBCR<n>_EL1 bits [31:0] are architecturally mapped to AArch64 System register [DBGBCR<n>_EL1\[31:0\]](#).

External register DBGBCR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGBCR<n>\[31:0\]](#).

DBGBCR<n>_EL1 is in the Core power domain.

If breakpoint n is not implemented then accesses to this register are:

- RES0 when IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalDebugAccess().
- A CONSTRAINED UNPREDICTABLE choice of RES0 or ERROR otherwise.

Attributes

DBGBCR<n>_EL1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|-----|----|-----|------|----|----|---|-----|---|---|------|-----|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | BT | | | | LBN | | | | SSC | | HMC | RES0 | | | | BAS | | | RES0 | PMC | E | | | |

When the E field is zero, all the other fields in the register are ignored.

Bits [31:24]

Reserved, RES0.

BT, bits [23:20]

Breakpoint Type. Possible values are:

| BT | Meaning |
|--------|---|
| 0b0000 | Unlinked instruction address match. DBGBVR<n>_EL1 is the address of an instruction. |
| 0b0001 | As 0b0000 but linked to a Context matching breakpoint. |
| 0b0010 | Unlinked Context ID match. When FEAT_VHE is implemented, EL2 is using AArch64, and the Effective value of HCR_EL2.E2H is 1, if either the PE is executing at EL0 with HCR_EL2.TGE set to 1 or the PE is executing at EL2, then DBGBVR<n>_EL1.ContextID must match the CONTEXTIDR_EL2 value. Otherwise, DBGBVR<n>_EL1.ContextID must match the CONTEXTIDR_EL1 value. |
| 0b0011 | As 0b0010, with linking enabled. |
| 0b0100 | Unlinked instruction address mismatch. DBGBVR<n>_EL1 is the address of an instruction to be stepped. |
| 0b0101 | As 0b0100, with linking enabled. |
| 0b0110 | Unlinked CONTEXTIDR_EL1 match. DBGBVR<n>_EL1.ContextID is a Context ID compared against CONTEXTIDR_EL1 . |
| 0b0111 | As 0b0110, with linking enabled. |
| 0b1000 | Unlinked VMID match. DBGBVR<n>_EL1.VMID is a VMID compared against VTTBR_EL2.VMID . |
| 0b1001 | As 0b1000, with linking enabled. |
| 0b1010 | Unlinked VMID and Context ID match. DBGBVR<n>_EL1.ContextID is a Context ID compared against CONTEXTIDR_EL1 , and DBGBVR<n>_EL1.VMID is a VMID compared against VTTBR_EL2.VMID . |
| 0b1011 | As 0b1010, with linking enabled. |
| 0b1100 | Unlinked CONTEXTIDR_EL2 match. DBGBVR<n>_EL1.ContextID2 is a Context ID compared against CONTEXTIDR_EL2 . |
| 0b1101 | As 0b1100, with linking enabled. |
| 0b1110 | Unlinked Full Context ID match. DBGBVR<n>_EL1.ContextID is compared against CONTEXTIDR_EL1 , and DBGBVR<n>_EL1.ContextID2 is compared against CONTEXTIDR_EL2 . |
| 0b1111 | As 0b1110, with linking enabled. |

Constraints on breakpoint programming mean some values are reserved under certain conditions.

For more information on the operation of the SSC, HMC, and PMC fields, and on the effect of programming this field to a reserved value, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions' and 'Reserved DBGBCR<n>_EL1.BT values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked breakpoint number. For Linked address matching breakpoints, this specifies the index of the Context-matching breakpoint linked to.

For all other breakpoint types this field is ignored and reads of the register return an UNKNOWN value.

This field is ignored when the value of DBGBCR<n>_EL1.E is 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the HMC and PMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information, including the effect of programming the fields to a reserved set of values, see 'Reserved DBGBCR<n>_EL1.{SSC, HMC, PMC} values'.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and PMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see [DBGBCR<n>_EL1](#).SSC description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [12:9]

Reserved, RES0.

BAS, bits [8:5]

When AArch32 is supported:

Byte address select. Defines which half-words an address-matching breakpoint matches, regardless of the instruction set and Execution state.

The permitted values depend on the breakpoint type.

For Address match breakpoints in either AArch32 or AArch64 state, the permitted values are:

| BAS | Match instruction at | Constraint for debuggers |
|--------|---|----------------------------------|
| 0b0011 | DBGBVR<n>_EL1 | Use for T32 instructions |
| 0b1100 | DBGBVR<n>_EL1 + 2 | Use for T32 instructions |
| 0b1111 | DBGBVR<n>_EL1 | Use for A64 and A32 instructions |

All other values are reserved.

For more information, see 'Using the BAS field in Address Match breakpoints'.

For Address mismatch breakpoints in an AArch32 stage 1 translation regime, the permitted values are:

| BAS | Match instruction at | Constraint for debuggers |
|--------|--|---|
| 0b0000 | - | Use for a match anywhere breakpoint |
| 0b0011 | DBGBVR<n>_EL1 | Use for stepping T32 instructions |
| 0b1100 | DBGBVR<n>_EL1 + 2 | Use for stepping T32 instructions |
| 0b1111 | DBGBVR<n>_EL1 | Use for stepping A64 and A32 instructions |

For more information, see 'Using the BAS field in Address Match breakpoints'.

For Context matching breakpoints, this field is RES1 and ignored.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

Bits [4:3]

Reserved, RES0.

PMC, bits [2:1]

Privilege mode control. Determines the Exception level or levels at which a Breakpoint debug event for breakpoint n is generated. This field must be interpreted along with the SSC and HMC fields, and there are constraints on the permitted values of the {HMC, SSC, PMC} fields. For more information see the [DBGBCR<n>_EL1.SSC](#) description.

For more information on the operation of the SSC, HMC, and PMC fields, see 'Execution conditions for which a breakpoint generates Breakpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable breakpoint [DBGVCR<n>_EL1](#). Possible values are:

| E | Meaning |
|-----|----------------------|
| 0b0 | Breakpoint disabled. |
| 0b1 | Breakpoint enabled. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBCR<n>_EL1

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

DBGBCR<n>_EL1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|------------------|---------------|
| Debug | 0x408 + (16 * n) | DBGBCR<n>_EL1 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

DBGBVR<n>_EL1, Debug Breakpoint Value Registers, n = 0 - 15

The DBGBVR<n>_EL1 characteristics are:

Purpose

Holds a virtual address, or a VMID and/or a context ID, for use in breakpoint matching. Forms breakpoint n together with control register [DBGBCR<n>_EL1](#).

Configuration

External register DBGBVR<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register [DBGBVR<n>_EL1\[63:0\]](#).

External register DBGBVR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGBVR<n>\[31:0\]](#).

External register DBGBVR<n>_EL1 bits [63:32] are architecturally mapped to AArch32 System register [DBGBXVR<n>\[31:0\]](#).

DBGBVR<n>_EL1 is in the Core power domain.

How this register is interpreted depends on the value of [DBGBCR<n>_EL1](#).BT.

- When [DBGBCR<n>_EL1](#).BT is 0b0x0x, this register holds a virtual address.
- When [DBGBCR<n>_EL1](#).BT is 0b001x, 0b011x, or 0b110x, this register holds a Context ID.
- When [DBGBCR<n>_EL1](#).BT is 0b100x, this register holds a VMID.
- When [DBGBCR<n>_EL1](#).BT is 0b101x, this register holds a VMID and a Context ID.
- When [DBGBCR<n>_EL1](#).BT is 0b111x, this register holds two Context ID values.

For other values of [DBGBCR<n>_EL1](#).BT, this register is RES0.

If breakpoint n is not implemented then accesses to this register are:

- RES0 when IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalDebugAccess().
- A CONSTRAINED UNPREDICTABLE choice of RES0 or ERROR otherwise.

Attributes

DBGBVR<n>_EL1 is a 64-bit register.

Field descriptions

When DBGBCR<n>_EL1.BT == 0b0x0x:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RESS[14:4] | | | | | | | | | | | Bits[52:49] | | | | VA[48:2] | | | | | | | | | | | | | | | | | |
| VA[48:2] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | RES0 | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

RESS[14:4], bits [63:53]

Reserved, Sign extended. Software must treat this field as RES0 if the most significant bit of VA is 0 or RES0, and as RES1 if the most significant bit of VA is 1.

Hardware always ignores the value of these bits and it is IMPLEMENTATION DEFINED whether:

- The bits are hardwired to a copy of the most significant bit of VA, meaning writes to these bits are ignored, and reads to the bits always return the hardwired value.
- The value in those bits can be written, and reads will return the last value written. The value held in those bits is ignored by hardware.

VA[52:49], bits [52:49]

When FEAT_LVA is implemented:

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Extension to RESS[14:4]. For more information, see RESS[14:4].

VA[48:2], bits [48:2]

If the address is being matched in an AArch64 stage 1 translation regime:

- This field contains bits[48:2] of the address for comparison.
- When FEAT_LVA is implemented, VA[52:49] forms the upper part of the address value. Otherwise, VA[52:49] are RESS.

If the address is being matched in an AArch32 stage 1 translation regime, the first 20 bits of this field are RES0, and the rest of the field contains bits[31:2] of the address for comparison.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT == 0b001x:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | ContextID | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:32]

Reserved, RES0.

ContextID, bits [31:0]

Context ID value for comparison.

The value is compared against [CONTEXTIDR_EL2](#) when (FEAT_VHE is implemented or FEAT_Debugv8p2 is implemented), EL2 is using AArch64, [HCR_EL2.E2H](#) is 1, and either:

- The PE is executing at EL2.
- [HCR_EL2.TGE](#) is 1, the PE is executing at EL0, and EL2 is enabled in the current Security state.

Otherwise, the value is compared against the following:

- [CONTEXTIDR](#) when the PE is executing at AArch32.
- [CONTEXTIDR_EL1](#) when the PE is executing at AArch64.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT == 0b011x, EL2 is implemented and (FEAT_VHE is implemented or FEAT_Debugv8p2 is implemented):

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | ContextID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

ContextID, bits [31:0]

Context ID value for comparison against [CONTEXTIDR_EL1](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT == 0b100x and EL2 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | VMID[15:8] | | | | | | | | VMID[7:0] | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

VMID[15:8], bits [47:40]

When FEAT_VHE is implemented and VTCR_EL2.VS == 1:

Extension to VMID[7:0]. For more information, see DBGVVR<n>_EL1.VMID[7:0].

If EL2 is using AArch32, this field is RES0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID[7:0], bits [39:32]

VMID value for comparison.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- [VTCR_EL2.VS](#) is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT == 0b101x and EL2 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | VMID[15:8] | | | | | | | | VMID[7:0] | | | | | | | |
| ContextID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:48]

Reserved, RES0.

VMID[15:8], bits [47:40]

When FEAT_VMID16 is implemented and VTCR_EL2.VS == 1:

Extension to VMID[7:0]. For more information, see DBGBCR<n>_EL1.VMID[7:0].

If EL2 is using AArch32, or if the implementation has an 8-bit VMID, this field is RES0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID[7:0], bits [39:32]

VMID value for comparison.

The VMID is 8 bits when any of the following are true:

- EL2 is using AArch32.
- VTCR_EL2.VS is 0.
- FEAT_VMID16 is not implemented.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

ContextID, bits [31:0]

Context ID value for comparison against CONTEXTIDR_EL1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When DBGBCR<n>_EL1.BT == 0b110x, EL2 is implemented and (FEAT_VHE is implemented or FEAT_Debugv8p2 is implemented):

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ContextID2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ContextID2, bits [63:32]

Context ID value for comparison against [CONTEXTIDR_EL2](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

Reserved, RES0.

When DBGBCR<n>_EL1.BT == 0b111x, EL2 is implemented and (FEAT_VHE is implemented or FEAT_Debugv8p2 is implemented):

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| | | | | | | | | | | | | | | | | | ContextID2 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | ContextID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

ContextID2, bits [63:32]

Context ID value for comparison against [CONTEXTIDR_EL2](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

ContextID, bits [31:0]

Context ID value for comparison against [CONTEXTIDR_EL1](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGBVR<n>_EL1

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

DBGBVR<n>_EL1 can be accessed through the external debug interface:

| Component | Offset | Instance | Range |
|-----------|------------------|---------------|-------|
| Debug | 0x400 + (16 * n) | DBGBVR<n>_EL1 | 63:0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

DBGCLAIMCLR_EL1, Debug CLAIM Tag Clear register

The DBGCLAIMCLR_EL1 characteristics are:

Purpose

Used by software to read the values of the CLAIM tag bits, and to clear CLAIM tag bits to 0.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the [DBGCLAIMSET_EL1](#) register.

Configuration

External register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch64 System register [DBGCLAIMCLR_EL1\[31:0\]](#).

External register DBGCLAIMCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGCLAIMCLR\[31:0\]](#).

DBGCLAIMCLR_EL1 is in the Core power domain.

An implementation must include eight CLAIM tag bits.

Attributes

DBGCLAIMCLR_EL1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RAZ/WI | | | | | | | | | | | | | | | | | | | | | | | | CLAIM | | | | | | | |

Bits [31:8]

Reserved, RAZ/WI.

CLAIM, bits [7:0]

Read or clear CLAIM tag bits. Reading this field returns the current value of the CLAIM tag bits.

Writing a 1 to one of these bits clears the corresponding CLAIM tag bit to 0. This is an indirect write to the CLAIM tag bits. A single write operation can clear multiple CLAIM tag bits to 0.

Writing 0 to one of these bits has no effect.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Accessing DBGCLAIMCLR_EL1

DBGCLAIMCLR_EL1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------------|
| Debug | 0xFA4 | DBGCLAIMCLR_EL1 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGCLAIMSET_EL1, Debug CLAIM Tag Set register

The DBGCLAIMSET_EL1 characteristics are:

Purpose

Used by software to set the CLAIM tag bits to 1.

The architecture does not define any functionality for the CLAIM tag bits.

Note

CLAIM tags are typically used for communication between the debugger and target software.

Used in conjunction with the [DBGCLAIMCLR_EL1](#) register.

Configuration

External register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch64 System register [DBGCLAIMSET_EL1\[31:0\]](#).

External register DBGCLAIMSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGCLAIMSET\[31:0\]](#).

DBGCLAIMSET_EL1 is in the Core power domain.

An implementation must include eight CLAIM tag bits.

Attributes

DBGCLAIMSET_EL1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RAZ/WI | | | | | | | | | | | | | | | | | | | | | | | | CLAIM | | | | | | | |

Bits [31:8]

Reserved, RAZ/WI.

CLAIM, bits [7:0]

Set CLAIM tag bits.

This field is RAO.

Writing a 1 to one of these bits sets the corresponding CLAIM tag bit to 1. This is an indirect write to the CLAIM tag bits. A single write operation can set multiple CLAIM tag bits to 1.

Writing 0 to one of these bits has no effect.

Accessing DBGCLAIMSET_EL1

DBGCLAIMSET_EL1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------------|
| Debug | 0xFA0 | DBGCLAIMSET_EL1 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRRX_EL0, Debug Data Transfer Register, Receive

The DBGDTRRX_EL0 characteristics are:

Purpose

Transfers data from an external debugger to the PE. For example, it is used by a debugger transferring commands and data to a debug target. See [DBGDTR_EL0](#) for additional architectural mappings. It is a component of the Debug Communications Channel.

Configuration

External register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to AArch64 System register [DBGDTRRX_EL0\[31:0\]](#).

External register DBGDTRRX_EL0 bits [31:0] are architecturally mapped to AArch32 System register [DBGDTRRXint\[31:0\]](#).

DBGDTRRX_EL0 is in the Core power domain.

Attributes

DBGDTRRX_EL0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Update DTRRX | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Update DTRRX.

Writes to this register:

- If RXfull is set to 1, set DTRRX to UNKNOWN.
- If RXfull is set to 0, update the value in DTRRX.

After the write, RXfull is set to 1.

Reads of this register:

- If RXfull is set to 1, return the last value written to DTRRX.
- If RXfull is set to 0, return an UNKNOWN value.

After the read, RXfull remains unchanged.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRRX_EL0

If [EDSCR](#).ITE == 0 when the PE exits Debug state on receiving a Restart request trigger event, the behavior of any operation issued by a DTR access in memory access mode that has not completed execution is **CONSTRAINED UNPREDICTABLE**, and must do one of the following:

- It must complete execution in Debug state before the PE executes the restart sequence.
- It must complete execution in Non-debug state before the PE executes the restart sequence.
- It must be abandoned. This means that the instruction does not execute. Any registers or memory accessed by the instruction are left in an UNKNOWN state.

DBGDTRRX_EL0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|--------------|
| Debug | 0x080 | DBGDTRRX_EL0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGDTRTX_EL0, Debug Data Transfer Register, Transmit

The DBGDTRTX_EL0 characteristics are:

Purpose

Transfers data from the PE to an external debugger. For example, it is used by a debug target to transfer data to the debugger. See [DBGDTR_EL0](#) for additional architectural mappings. It is a component of the Debug Communication Channel.

Configuration

External register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to AArch64 System register [DBGDTRTX_EL0\[31:0\]](#).

External register DBGDTRTX_EL0 bits [31:0] are architecturally mapped to AArch32 System register [DBGDTRTXint\[31:0\]](#).

DBGDTRTX_EL0 is in the Core power domain.

Attributes

DBGDTRTX_EL0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Return DTRTX | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Return DTRTX.

Reads of this register:

- If TXfull is set to 1, return the last value written to DTRTX.
- If TXfull is set to 0, return an UNKNOWN value.

After the read, TXfull is cleared to 0.

Writes to this register:

- If TXfull is set to 1, set DTRTX to UNKNOWN.
- If TXfull is set to 0, update the value in DTRTX.

After the write, TXfull remains unchanged.

For the full behavior of the Debug Communications Channel, see 'The Debug Communication Channel and Instruction Transfer Register'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGDTRTX_EL0

If [EDSCR](#).ITE == 0 when the PE exits Debug state on receiving a Restart request trigger event, the behavior of any operation issued by a DTR access in memory access mode that has not completed execution is CONstrained UNPREDICTABLE, and must do one of the following:

- It must complete execution in Debug state before the PE executes the restart sequence.
- It must complete execution in Non-debug state before the PE executes the restart sequence.
- It must be abandoned. This means that the instruction does not execute. Any registers or memory accessed by the instruction are left in an UNKNOWN state.

DBGDTRTX_EL0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|--------------|
| Debug | 0x08C | DBGDTRTX_EL0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

Purpose

Configuration

Attributes

Field descriptions

Page 3987

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [23:21]

Reserved, RES0.

WT, bit [20]

Watchpoint type. Possible values are:

| WT | Meaning |
|-----|------------------------------|
| 0b0 | Unlinked data address match. |
| 0b1 | Linked data address match. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LBN, bits [19:16]

Linked breakpoint number. For Linked data address watchpoints, this specifies the index of the Context-matching breakpoint linked to.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

SSC, bits [15:14]

Security state control. Determines the Security states under which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the HMC and PAC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

HMC, bit [13]

Higher mode control. Determines the debug perspective for deciding when a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and PAC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

BAS, bits [12:5]

Byte address select. Each bit of this field selects whether a byte from within the word or double-word addressed by [DBGWVR<n>_EL1](#) is being watched.

| BAS | Description |
|----------|---|
| xxxxxx1 | Match byte at DBGWVR<n>_EL1 |
| xxxxxx1x | Match byte at DBGWVR<n>_EL1 + 1 |
| xxxxx1xx | Match byte at DBGWVR<n>_EL1 + 2 |
| xxxx1xxx | Match byte at DBGWVR<n>_EL1 + 3 |

In cases where [DBGWVR<n>_EL1](#) addresses a double-word:

| BAS | Description, if DBGWVR<n>_EL1[2] == 0 |
|----------|---|
| xxx1xxxx | Match byte at DBGWVR<n>_EL1 + 4 |
| xx1xxxxx | Match byte at DBGWVR<n>_EL1 + 5 |
| x1xxxxxx | Match byte at DBGWVR<n>_EL1 + 6 |
| 1xxxxxxx | Match byte at DBGWVR<n>_EL1 + 7 |

If [DBGWVR<n>_EL1\[2\]](#) == 1, only BAS[3:0] is used. Arm deprecates setting [DBGWVR<n>_EL1\[2\]](#) == 1.

The valid values for BAS are non-zero binary number all of whose set bits are contiguous. All other values are reserved and must not be used by software. See 'Reserved DBGWCR<n>.BAS values'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

LSC, bits [4:3]

Load/store control. This field enables watchpoint matching on the type of access being made. Possible values of this field are:

| LSC | Meaning |
|------|---|
| 0b01 | Match instructions that load from a watchpointed address. |
| 0b10 | Match instructions that store to a watchpointed address. |
| 0b11 | Match instructions that load from or store to a watchpointed address. |

All other values are reserved, but must behave as if the watchpoint is disabled. Software must not rely on this property as the behavior of reserved values might change in a future revision of the architecture.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PAC, bits [2:1]

Privilege of access control. Determines the Exception level or levels at which a Watchpoint debug event for watchpoint n is generated. This field must be interpreted along with the SSC and HMC fields.

For more information on the operation of the SSC, HMC, and PAC fields, see 'Execution conditions for which a watchpoint generates Watchpoint exceptions'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E, bit [0]

Enable watchpoint n. Possible values are:

| E | Meaning |
|-----|----------------------|
| 0b0 | Watchpoint disabled. |
| 0b1 | Watchpoint enabled. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing DBGWCR<n>_EL1

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

DBGWCR<n>_EL1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|------------------|---------------|
| Debug | 0x808 + (16 * n) | DBGWCR<n>_EL1 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

DBGWVR<n>_EL1, Debug Watchpoint Value Registers, n = 0 - 15

The DBGWVR<n>_EL1 characteristics are:

Purpose

Holds a data address value for use in watchpoint matching. Forms watchpoint n together with control register [DBGWCR<n>_EL1](#).

Configuration

External register DBGWVR<n>_EL1 bits [63:0] are architecturally mapped to AArch64 System register [DBGWVR<n>_EL1\[63:0\]](#).

External register DBGWVR<n>_EL1 bits [31:0] are architecturally mapped to AArch32 System register [DBGWVR<n>\[31:0\]](#).

DBGWVR<n>_EL1 is in the Core power domain.

If watchpoint n is not implemented then accesses to this register are:

- When IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalDebugAccess(), RES0.
- Otherwise, a CONSTRAINED UNPREDICTABLE choice of RES0 or ERROR.

Attributes

DBGWVR<n>_EL1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|-------------|----|----|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RESS[14:4] | | | | | | | | | | | Bits[52:49] | | | VA[48:2] | | | | | | | | | | | | | | | | | | |
| VA[48:2] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | RES0 | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

RESS[14:4], bits [63:53]

Reserved, Sign extended. Hardware and software must treat this field as RES0 if the most significant bit of VA is 0 or RES0, and as RES1 if the most significant bit of VA is 1.

Hardware always ignores the value of these bits and it is IMPLEMENTATION DEFINED whether:

- The bits are hardwired to a copy of the most significant bit of VA, meaning writes to these bits are ignored, and reads to the bits always return the hardwired value.
- The value in those bits can be written, and reads will return the last value written. The value held in those bits is ignored by hardware.

VA[52:49], bits [52:49]

When FEAT_LVA is implemented:

Extension to VA[48:2]. For more information, see VA[48:2].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Extension to RESS[14:4]. For more information, see RESS[14:4].

VA[48:2], bits [48:2]

Bits[48:2] of the address value for comparison.

When FEAT_LVA is implemented, VA[52:49] forms the upper part of the address value. Otherwise, VA[52:49] are RESS.

Arm deprecates setting [DBGWVR<n>_EL1\[2\] == 1](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

Accessing DBGWVR<n>_EL1**Note**

SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

DBGWVR<n>_EL1 can be accessed through the external debug interface:

| Component | Offset | Instance | Range |
|-----------|------------------|---------------|-------|
| Debug | 0x800 + (16 * n) | DBGWVR<n>_EL1 | 63:0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalDebugAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

EDAA32PFR, External Debug Auxiliary Processor Feature Register

The EDAA32PFR characteristics are:

Purpose

Provides information about implemented PE features.

Note

The register mnemonic, EDAA32PFR, is derived from previous definitions of this register that defined this register only when AArch64 was not supported.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

It is IMPLEMENTATION DEFINED whether EDAA32PFR is implemented in the Core power domain or in the Debug power domain.

Attributes

EDAA32PFR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|-----|----|----|----|-----|----|----|----|------|----|----|----|------|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | MSA_frac | | | | EL3 | | | | EL2 | | | | PMSA | | | | VMSA | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

Bits [63:20]

Reserved, RES0.

MSA_frac, bits [19:16]

When EDAA32PFR.PMSA == 0b0000 and EDAA32PFR.VMSA == 0b1111:

Memory System Architecture fractional field. This holds the information on additional Memory System Architectures supported. Defined values are:

| MSA_frac | Meaning |
|----------|---|
| 0b0001 | PMSAv8-64 supported in all translation regimes. VMSAv8-64 not supported. |
| 0b0010 | PMSAv8-64 supported in all translation regimes. In addition to PMSAv8-64, stage 1 EL1&0 translation regime also supports VMSAv8-64. |

All other values are reserved.

Otherwise:

Reserved, RES0.

EL3, bits [15:12]**When EDPFR.EL3 == 0b0000:**

AArch32 EL3 Exception level handling. Defined values are:

| EL3 | Meaning |
|--------|---|
| 0b0000 | EL3 is not implemented or can be executed in AArch64 state. |
| 0b0001 | EL3 can be executed in AArch32 state only. |

All other values are reserved.

Note

[EDPFR](#).{EL1, EL0} indicate whether EL1 and EL0 can only be executed in AArch32 state.

Otherwise:

Reserved, RAZ.

EL2, bits [11:8]**When EDPFR.EL2 == 0b0000:**

AArch32 EL2 Exception level handling. Defined values are:

| EL2 | Meaning |
|--------|---|
| 0b0000 | EL2 is not implemented or can be executed in AArch64 state. |
| 0b0001 | EL2 can be executed in AArch32 state only. |

All other values are reserved.

Note

[EDPFR](#).{EL1, EL0} indicate whether EL1 and EL0 can only be executed in AArch32 state.

Otherwise:

Reserved, RAZ.

PMSA, bits [7:4]

Indicates support for a 32-bit PMSA. Defined values are:

| PMSA | Meaning |
|--------|------------------------|
| 0b0000 | PMSA-32 not supported. |
| 0b0100 | PMSAv8-32 supported. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

VMSA, bits [3:0]**When EDAA32PFR.PMSA != 0b0000:**

Indicates support for a VMSA in addition to a 32-bit PMSA. Defined values are:

| VMSA | Meaning |
|--------|---------------------|
| 0b0000 | VMSA not supported. |

All other values are reserved.

When EDAA32PFR.PMSA == 0b0000:

Defined values are:

| VMSA | Meaning |
|--------|---|
| 0b0000 | VMSAv8-64 supported. |
| 0b1111 | Memory system architecture described by EDAA32PFR.MSA_frac. |

All other values are reserved.

In Armv8-A, the only permitted value is 0b0000.

Otherwise:

Reserved, RAZ.

Accessing EDAA32PFR

EDAA32PFR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| Debug | 0xD60 | EDAA32PFR |

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register are **IMPDEF**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDACR, External Debug Auxiliary Control Register

The EDACR characteristics are:

Purpose

Allows implementations to support IMPLEMENTATION DEFINED controls.

Configuration

It is IMPLEMENTATION DEFINED whether EDACR is implemented in the Core power domain or in the Debug power domain.

If FEAT_DoPD is implemented, this register is implemented in the Core power domain.

If FEAT_DoPD is not implemented, the power domain that this register is implemented in is IMPLEMENTATION DEFINED.

Changing this register from its reset value causes IMPLEMENTATION DEFINED behavior, including possible deviation from the architecturally-defined behavior.

If the EDACR contains any control bits that must be preserved over power down, then these bits must be accessible by the external debug interface when the OS Lock is locked, [OSLSR_EL1](#).OSLK == 1, and when the Core is powered off.

Attributes

EDACR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The following resets apply:

- If the register is implemented in the Core power domain:
 - On a Cold reset, this field resets to an architecturally UNKNOWN value.
 - On an External debug reset, the value of this field is unchanged.
 - On a Warm reset, the value of this field is unchanged.
- If the register is implemented in the External debug power domain:
 - On a Cold reset, the value of this field is unchanged.
 - On an External debug reset, this field resets to an architecturally UNKNOWN value.
 - On a Warm reset, the value of this field is unchanged.

Accessing EDACR

EDACR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
|-----------|--------|----------|

| | | |
|-------|-------|-------|
| Debug | 0x094 | EDACR |
|-------|-------|-------|

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register are **IMPDEF**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDCIDR0, External Debug Component Identification Register 0

The EDCIDR0 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Component Identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

EDCIDR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL_0 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Preamble.

Reads as 0x0D.

Access to this field is **RO**.

Accessing EDCIDR0

EDCIDR0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xFF0 | EDCIDR0 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

EDCIDR1, External Debug Component Identification Register 1

The EDCIDR1 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Component Identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

EDCIDR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | CLASS | | | PRMBL_1 | | | | |

Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

| CLASS | Meaning |
|--------|----------------------|
| 0b1001 | CoreSight component. |

Other values are defined by the CoreSight Architecture.

This field reads as 0x9.

PRMBL_1, bits [3:0]

Preamble.

Reads as 0b0000.

Access to this field is **RO**.

Accessing EDCIDR1

EDCIDR1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xFF4 | EDCIDR1 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDCIDR2, External Debug Component Identification Register 2

The EDCIDR2 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Component Identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

EDCIDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL_2 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Preamble.

Reads as 0x05.

Access to this field is **RO**.

Accessing EDCIDR2

EDCIDR2 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xFF8 | EDCIDR2 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

EDCIDR3, External Debug Component Identification Register 3

The EDCIDR3 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Component Identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

EDCIDR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL_3 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Preamble.

Reads as 0xB1.

Access to this field is **RO**.

Accessing EDCIDR3

EDCIDR3 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xFFC | EDCIDR3 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

EDCIDSR, External Debug Context ID Sample Register

The EDCIDSR characteristics are:

Purpose

Contains the sampled value of the Context ID, captured on reading [EDPCSR](#)[31:0].

Configuration

EDCIDSR is in the Core power domain.

This register is present only when FEAT_PCSRv8 is implemented and FEAT_PCSRv8p2 is not implemented. Otherwise, direct accesses to EDCIDSR are RES0.

Implemented only if the OPTIONAL PC Sample-based Profiling Extension is implemented in the external debug registers space.

Note

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors registers space.

Attributes

EDCIDSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CONTEXTIDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

CONTEXTIDR, bits [31:0]

Context ID. The value of CONTEXTIDR that is associated with the most recent [EDPCSR](#) sample. When the most recent [EDPCSR](#) sample was generated:

- If EL1 is using AArch64, then the Context ID is sampled from [CONTEXTIDR_EL1](#).
- If EL1 is using AArch32, then the Context ID is sampled from [CONTEXTIDR](#).
- If EL3 is implemented and is using AArch32, then [CONTEXTIDR](#) is a banked register, and EDCIDSR samples the current banked copy of [CONTEXTIDR](#) for the Security state that is associated with the most recent [EDPCSR](#) sample.

Because the value written to EDCIDSR is an indirect read of CONTEXTIDR, it is CONSTRAINED UNPREDICTABLE whether EDCIDSR is set to the original or new value if [EDPCSR](#) samples:

- An instruction that writes to CONTEXTIDR.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing EDCIDSR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

EDCIDSR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0x0A4 | EDCIDSR |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDDEVAFF0, External Debug Device Affinity register 0

The EDDEVAFF0 characteristics are:

Purpose

Copy of the low half of the PE [MPIDR_EL1](#) register that allows a debugger to determine which PE in a multiprocessor system the external debug component relates to.

Configuration

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

EDDEVAFF0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| MPIDR_EL1lo | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

MPIDR_EL1lo, bits [31:0]

[MPIDR_EL1](#) low half. Read-only copy of the low half of [MPIDR_EL1](#), as seen from the highest implemented Exception level.

Accessing EDDEVAFF0

EDDEVAFF0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| Debug | 0xFA8 | EDDEVAFF0 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

EDDEVAFF1, External Debug Device Affinity register 1

The EDDEVAFF1 characteristics are:

Purpose

Copy of the high half of the PE [MPIDR_EL1](#) register that allows a debugger to determine which PE in a multiprocessor system the external debug component relates to.

Configuration

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

EDDEVAFF1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------------------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | MPIDR_EL1hi | | | | | | | | | | | | | | | |

MPIDR_EL1hi, bits [31:0]

[MPIDR_EL1](#) high half. Read-only copy of the high half of [MPIDR_EL1](#), as seen from the highest implemented Exception level.

Accessing EDDEVAFF1

EDDEVAFF1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| Debug | 0xFAC | EDDEVAFF1 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

EDDEVARCH, External Debug Device Architecture register

The EDDEVARCH characteristics are:

Purpose

Identifies the programmers' model architecture of the external debug component.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain.

If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

EDDEVARCH is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|---------|----------|----|----|----|---------|----|----|----|----|----------|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ARCHITECT | | | | | | | | | | | PRESENT | REVISION | | | | ARCHVER | | | | | ARCHPART | | | | | | | | | | |

ARCHITECT, bits [31:21]

Defines the architecture of the component. For debug, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, 0x4.

Bits [27:21] are the JEP106 ID code, 0x3B.

Reads as 0b01000111011.

Access to this field is **RO**.

PRESENT, bit [20]

Indicates that the DEVARCH is present.

Reads as 0b1.

Access to this field is **RO**.

REVISION, bits [19:16]

Defines the architecture revision. For architectures defined by Arm this is the minor revision.

For debug, the revision defined by Armv8 is 0x0.

All other values are reserved.

Reads as 0b0000.

Access to this field is **RO**.

ARCHVER, bits [15:12]

Architecture Version. Defines the architecture version of the component. Defined values are:

| ARCHVER | Meaning |
|----------------|---|
| 0b0110 | Armv8 debug architecture. |
| 0b0111 | Armv8 debug architecture with Virtualization Host Extensions. |
| 0b1000 | Armv8.2 debug architecture, FEAT_Debugv8p2. |
| 0b1001 | Armv8.4 debug architecture, FEAT_Debugv8p4. |
| 0b1010 | Armv8.8 debug architecture, FEAT_Debugv8p8. |

EDDEVARCH.ARCHVER and EDDEVARCH.ARCHPART are also defined as a single field, EDDEVARCH.ARCHID, so that EDDEVARCH.ARCHVER is EDDEVARCH.ARCHID[15:12].

FEAT_VHE adds the functionality identified by the value 0b0111.

FEAT_Debugv8p2 adds the functionality identified by the value 0b1000.

FEAT_Debugv8p4 adds the functionality identified by the value 0b1001.

FEAT_Debugv8p8 adds the functionality identified by the value 0b1010.

From Armv8.1, when FEAT_VHE is implemented the value 0b0110 is not permitted.

From Armv8.2, the values 0b0110 and 0b0111 are not permitted.

From Armv8.4, the value 0b1000 is not permitted.

From Armv8.8, the value 0b1001 is not permitted.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

EDDEVARCH.ARCHVER and EDDEVARCH.ARCHPART are also defined as a single field, EDDEVARCH.ARCHID, so that EDDEVARCH.ARCHPART is EDDEVARCH.ARCHID[11:0].

Armv8-A debug architecture.

Reads as 0xA15.

Access to this field is **RO**.

Accessing EDDEVARCH

EDDEVARCH can be accessed through the external debug interface:

| Component | Offset | Instance |
|------------------|---------------|-----------------|
| Debug | 0xFBC | EDDEVARCH |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

EDDEVID, External Debug Device ID register 0

The EDDEVID characteristics are:

Purpose

Provides extra information for external debuggers about features of the debug implementation.

Configuration

If FEAT_DoPD is implemented, this register is in the Core power domain.

If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

EDDEVID is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|---------|----|----|----|------|----|----|----|----|----|----|----|------------|----|----|----|----------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | AuxRegs | | | | RES0 | | | | | | | | DebugPower | | | | PCSample | | | | | | | | | | | |

Bits [31:28]

Reserved, RES0.

AuxRegs, bits [27:24]

Indicates support for Auxiliary registers. Defined values are:

| AuxRegs | Meaning |
|---------|--|
| 0b0000 | None supported. |
| 0b0001 | Support for External Debug Auxiliary Control Register, EDACR . |

All other values are reserved.

Bits [23:8]

Reserved, RES0.

DebugPower, bits [7:4]

Indicates support for the FEAT_DoPD feature. Defined values are:

| DebugPower | Meaning |
|------------|---|
| 0b0000 | FEAT_DoPD not implemented. Registers in the external debug interface register map are implemented in a mix of the Debug and Core power domains. |
| 0b0001 | FEAT_DoPD implemented. All registers in the external debug interface register map are implemented in the Core power domain. |

FEAT_DoPD implements the functionality added by the value 0b0001.

All other values are reserved.

PCSample, bits [3:0]

Indicates the level of PC Sample-based Profiling support using external debug registers. Defined values are:

| PCSample | Meaning |
|----------|---|
| 0b0000 | PC Sample-based Profiling Extension is not implemented in the external debug registers space. |
| 0b0010 | Only EDPCSR and EDCIDS are implemented. This option is only permitted if EL3 and EL2 are not implemented. |
| 0b0011 | EDPCSR , EDCIDS , and EDVIDSR are implemented. |

All other values are reserved.

When FEAT_PCSRv8p2 is implemented, the only permitted value is 0b0000.

Note

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of [PMDEVID.PCSample](#).

Accessing EDDEVID

EDDEVID can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xFC8 | EDDEVID |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

EDDEVID1, External Debug Device ID register 1

The EDDEVID1 characteristics are:

Purpose

Provides extra information for external debuggers about features of the debug implementation.

Configuration

If FEAT_DoPD is implemented, this register is in the Core power domain.

If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

EDDEVID1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | PCSROffset | | | |

Bits [31:4]

Reserved, RES0.

PCSROffset, bits [3:0]

This field indicates the offset applied to PC samples returned by reads of [EDPCSR](#). Permitted values of this field in Armv8 are:

| PCSROffset | Meaning |
|------------|--|
| 0b0000 | EDPCSR not implemented. |
| 0b0010 | EDPCSR implemented, and samples have no offset applied and do not sample the instruction set state in AArch32 state. |

When FEAT_PCSRv8p2 is implemented, the only permitted value is 0b0000.

Note

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors register space, as indicated by the value of [PMDEVID](#).PCSample.

Accessing EDDEVID1

EDDEVID1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xFC4 | EDDEVID1 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDDEVID2, External Debug Device ID register 2

The EDDEVID2 characteristics are:

Purpose

Reserved for future descriptions of features of the debug implementation.

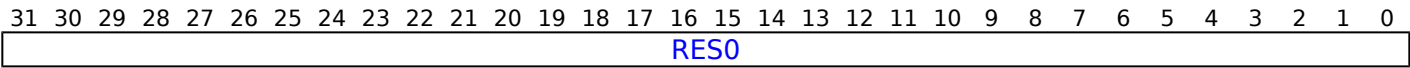
Configuration

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

EDDEVID2 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing EDDEVID2

EDDEVID2 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xFC0 | EDDEVID2 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

EDDEVTTYPE, External Debug Device Type register

The EDDEVTTYPE characteristics are:

Purpose

Indicates to a debugger that this component is part of a PEs debug logic.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

EDDEVTTYPE is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-----|---|---|---|-------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | SUB | | | | MAJOR | | | |

Bits [31:8]

Reserved, RES0.

SUB, bits [7:4]

Subtype. Indicates this is a component within a PE.

Reads as 0b0001.

Access to this field is **RO**.

MAJOR, bits [3:0]

Major type. Indicates this is a debug logic component.

Reads as 0b0101.

Access to this field is **RO**.

Accessing EDDEVTTYPE

EDDEVTTYPE can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|------------|
| Debug | 0xFCC | EDDEVTTYPE |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

EDDFR, External Debug Feature Register

The EDDFR characteristics are:

Purpose

Provides top level information about the debug system.

Note

Debuggers must use [EDDEVARCH](#) to determine the Debug architecture version.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

It is IMPLEMENTATION DEFINED whether EDDFR is implemented in the Core power domain or in the Debug power domain.

Attributes

EDDFR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|------|----|----|----|------|----|----|----|-----------|----|----|----|------|----|----|----|---------|----|----|----|----------|----|----|----|---------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | TraceFilt | | | | | | | | UNKNOWN | | | | | | | | | | | |
| CTX_CMPs | | | | RES0 | | | | WRPs | | | | RES0 | | | | BRPs | | | | PMUVer | | | | TraceVer | | | | UNKNOWN | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:44]

Reserved, RES0.

TraceFilt, bits [43:40]

Armv8.4 Self-hosted Trace Extension version. Defined values are:

| TraceFilt | Meaning |
|-----------|---|
| 0b0000 | Armv8.4 Self-hosted Trace Extension is not implemented. |
| 0b0001 | Armv8.4 Self-hosted Trace Extension is implemented. |

All other values are reserved.

FEAT_TRF implements the functionality added by 0b0001.

From Armv8.4, the permitted values are 0b0000 and 0b0001.

Bits [39:32]

Reserved, UNKNOWN.

CTX_CMPs, bits [31:28]

Number of breakpoints that are context-aware, minus 1. These are the highest numbered breakpoints.

In an Armv8-A implementation that supports AArch64, this field returns the value of [ID_AA64DFR0_EL1](#).CTX_CMPs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [27:24]

Reserved, RES0.

WRPs, bits [23:20]

Number of watchpoints, minus 1. The value of 0b0000 is reserved.

In an Armv8-A implementation that supports AArch64, this field returns the value of [ID_AA64DFR0_EL1](#).WRPs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [19:16]

Reserved, RES0.

BRPs, bits [15:12]

Number of breakpoints, minus 1. The value of 0b0000 is reserved.

In an Armv8-A implementation that supports AArch64, this field returns the value of [ID_AA64DFR0_EL1](#).BRPs.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PMUVer, bits [11:8]

Performance Monitors Extension version.

This field does not follow the standard ID scheme, but uses the alternative ID scheme described in 'Alternative ID scheme used for the Performance Monitors Extension version'

Defined values are:

| PMUVer | Meaning |
|--------|--|
| 0b0000 | Performance Monitors Extension not implemented. |
| 0b0001 | Performance Monitors Extension, PMUv3 implemented. |
| 0b0100 | PMUv3 for Armv8.1. As 0b0001, and adds support for: <ul style="list-style-type: none"> Extended 16-bit PMEVTYPER<n>_EL0.evtCount field. If EL2 is implemented, the MDCR_EL2.HPMD control. |
| 0b0101 | PMUv3 for Armv8.4. As 0b0100, and adds support for the PMMIR_EL1 register. |
| 0b0110 | PMUv3 for Armv8.5. As 0b0101, and adds support for: <ul style="list-style-type: none"> 64-bit event counters. If EL2 is implemented, the MDCR_EL2.HCCD control. If EL3 is implemented, the MDCR_EL3.SCCD control. |
| 0b0111 | PMUv3 for Armv8.7. As 0b0110, and adds support for: <ul style="list-style-type: none"> The PMCR_EL0.FZO and, if EL2 is implemented, MDCR_EL2.HPMFZO controls. If EL3 is implemented, the MDCR_EL3.{MPMX,MCCD} controls. |
| 0b1000 | PMUv3 for Armv8.8. As 0b0111, and: <ul style="list-style-type: none"> Extends the Common event number space to include 0x0040 to 0x00BF and 0x4040 to 0x40BF. Removes the CONSTRAINED UNPREDICTABLE behaviors if a reserved or unimplemented PMU event number is selected. |
| 0b1111 | IMPLEMENTATION DEFINED form of performance monitors supported, PMUv3 not supported. Arm does not recommend this value for new implementations. |

All other values are reserved.

FEAT_PMUv3 implements the functionality identified by the value 0b0001.

FEAT_PMUv3p1 implements the functionality identified by the value 0b0100.

FEAT_PMUv3p4 implements the functionality identified by the value 0b0101.

FEAT_PMUv3p5 implements the functionality identified by the value 0b0110.

FEAT_PMUv3p7 implements the functionality identified by the value 0b0111.

FEAT_PMUv3p8 implements the functionality identified by the value 0b1000.

From Armv8.1, if FEAT_PMUv3 is implemented, the value 0b0001 is not permitted.

From Armv8.4, if FEAT_PMUv3 is implemented, the value 0b0100 is not permitted.

From Armv8.5, if FEAT_PMUv3 is implemented, the value 0b0101 is not permitted.

From Armv8.7, if FEAT_PMUv3 is implemented, the value 0b0110 is not permitted.

From Armv8.8, if FEAT_PMUv3 is implemented, the value 0b0111 is not permitted.

TraceVer, bits [7:4]

Trace support. Indicates whether System register interface to a PE trace unit is implemented. Defined values are:

| TraceVer | Meaning |
|----------|---|
| 0b0000 | PE trace unit System registers not implemented. |
| 0b0001 | PE trace unit System registers implemented. |

All other values are reserved.

A value of 0b0000 only indicates that no System register interface to a PE trace unit is implemented. A PE trace unit might nevertheless be implemented without a System register interface.

In an Armv8-A implementation that supports AArch64, this field returns the value of [ID_AA64DFR0_EL1](#).TraceVer.

Bits [3:0]

Reserved, UNKNOWN.

Accessing EDDFR**EDDFR can be accessed through the external debug interface:**

| Component | Offset | Instance | Range |
|-----------|--------|----------|-------|
| Debug | 0xD28 | EDDFR | 31:0 |

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register are **IMPDEF**.

| Component | Offset | Instance | Range |
|-----------|--------|----------|-------|
| Debug | 0xD2C | EDDFR | 63:32 |

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register are **IMPDEF**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDECCR, External Debug Exception Catch Control Register

The EDECCR characteristics are:

Purpose

Controls Exception Catch debug events. For more information, see 'Summary of Exception Catch debug event control'.

Configuration

External register EDECCR bits [31:0] are architecturally mapped to AArch64 System register [OSECCR_EL1\[31:0\]](#).

External register EDECCR bits [31:0] are architecturally mapped to AArch32 System register [DBGOSECCR\[31:0\]](#).

EDECCR is in the Core power domain.

Attributes

EDECCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|------|------|------|------|------|------|------|------|------|------|------|-----|-----|-----|-----|------|------|------|------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 |
| RES0 | | | | | | | | | RLR2 | RLR1 | RLR0 | RES0 | RLE2 | RLE1 | RLE0 | NSR3 | NSR2 | NSR1 | NSR0 | SR3 | SR2 | SR1 | SR0 | NSE3 | NSE2 | NSE1 | NSE0 |

Bits [31:23]

Reserved, RES0.

RLR2, bit [22]

When FEAT_RME is implemented:

Controls exception catch on exception return to Realm EL2 in conjunction with EDECCR.RLE2.

| RLR2 | Meaning |
|------|---|
| 0b0 | If EDECCR.RLE2 is 0, then Exception Catch debug events are disabled for Realm EL2. If EDECCR.RLE2 is 1, then Exception Catch debug events are enabled for exception entry and exception return to Realm EL2. |
| 0b1 | If EDECCR.RLE2 is 0, then Exception Catch debug events are enabled for exception returns to Realm EL2. If EDECCR.RLE2 is 1, then Exception Catch debug events are enabled for exception entry to Realm EL2. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

RLR1, bit [21]**When FEAT_RME is implemented:**

Controls exception catch on exception return to Realm EL1 in conjunction with EDECCR.RLE1.

| RLR1 | Meaning |
|-------------|---|
| 0b0 | If EDECCR.RLE1 is 0, then Exception Catch debug events are disabled for Realm EL1. If EDECCR.RLE1 is 1, then Exception Catch debug events are enabled for exception entry and exception return to Realm EL1. |
| 0b1 | If EDECCR.RLE1 is 0, then Exception Catch debug events are enabled for exception returns to Realm EL1. If EDECCR.RLE1 is 1, then Exception Catch debug events are enabled for exception entry to Realm EL1. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

RLR0, bit [20]**When FEAT_RME is implemented:**

Controls exception catch on exception return to Realm EL0.

| RLR0 | Meaning |
|-------------|--|
| 0b0 | Exception Catch debug events are disabled for Realm EL0. |
| 0b1 | Exception Catch debug events are enabled for exception returns to Realm EL0. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Bit [19]

Reserved, RES0.

RLE2, bit [18]**When FEAT_RME is implemented:**

Controls exception catch on exception entry to Realm EL2. Also controls exception catch on exception return to Realm EL2 in conjunction with EDECCR.RLR2.

| RLE2 | Meaning |
|-------------|---|
| 0b0 | If EDECCR.RLR2 is 0, then Exception Catch debug events are disabled for Realm EL2. If EDECCR.RLR2 is 1, then Exception Catch debug events are enabled for exception returns to Realm EL2. |
| 0b1 | If EDECCR.RLR2 is 0, then Exception Catch debug events are enabled for exception entry and exception return to Realm EL2. If EDECCR.RLR2 is 1, then Exception Catch debug events are enabled for exception entry to Realm EL2. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

RLE1, bit [17]

When FEAT_RME is implemented:

Controls exception catch on exception entry to Realm EL1. Also controls exception catch on exception return to Realm EL1 in conjunction with EDECCR.RLR1.

| RLE1 | Meaning |
|-------------|---|
| 0b0 | If EDECCR.RLR1 is 0, then Exception Catch debug events are disabled for Realm EL1. If EDECCR.RLR1 is 1, then Exception Catch debug events are enabled for exception returns to Realm EL1. |
| 0b1 | If EDECCR.RLR1 is 0, then Exception Catch debug events are enabled for exception entry and exception return to Realm EL1. If EDECCR.RLR1 is 1, then Exception Catch debug events are enabled for exception entry to Realm EL1. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

RLE0, bit [16]

Access to this field is **RES0**.

NSR3, bit [15]

Access to this field is **RES0**.

NSR2, bit [14]

When FEAT_Debugv8p2 is implemented and Non-secure EL2 is implemented:

Controls exception catch on exception return to Non-secure EL2 in conjunction with EDECCR.NSE2.

| NSR2 | Meaning |
|-------------|--|
| 0b0 | If EDECCR.NSE2 is 0, then Exception Catch debug events are disabled for Non-secure EL2. If EDECCR.NSE2 is 1, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Non-secure EL2. |
| 0b1 | If EDECCR.NSE2 is 0, then Exception Catch debug events are enabled for exception returns to Non-secure EL2. If EDECCR.NSE2 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Non-secure EL2. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

NSR1, bit [13]

When FEAT_Debugv8p2 is implemented and Non-secure EL1 is implemented:

Controls exception catch on exception return to Non-secure EL1 in conjunction with EDECCR.NSE1.

| NSR1 | Meaning |
|------|--|
| 0b0 | If EDECCR.NSE1 is 0, then Exception Catch debug events are disabled for Non-secure EL1. If EDECCR.NSE1 is 1, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Non-secure EL1. |
| 0b1 | If EDECCR.NSE1 is 0, then Exception Catch debug events are enabled for exception returns to Non-secure EL1. If EDECCR.NSE1 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Non-secure EL1. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

NSR0, bit [12]

When FEAT_Debugv8p2 is implemented and Non-secure EL0 is implemented:

Controls exception catch on exception return to Non-secure EL0.

| NSR0 | Meaning |
|------|---|
| 0b0 | Exception Catch debug events are disabled for Non-secure EL0. |
| 0b1 | Exception Catch debug events are enabled for exception returns to Non-secure EL0. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SR3, bit [11]

When FEAT_Debugv8p2 is implemented and EL3 is implemented:

Controls exception catch on exception return to EL3 in conjunction with EDECCR.SE3.

| SR3 | Meaning |
|------------|--|
| 0b0 | If EDECCR.SE3 is 0, then Exception Catch debug events are disabled for EL3. If EDECCR.SE3 is 1, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to EL3. |
| 0b1 | If EDECCR.SE3 is 0, then Exception Catch debug events are enabled for exception returns to EL3. If EDECCR.SE3 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to EL3. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SR2, bit [10]

When FEAT_Debugv8p2 is implemented and FEAT_SEL2 is implemented:

Controls exception catch on exception return to Secure EL2 in conjunction with EDECCR.SE2.

| SR2 | Meaning |
|------------|--|
| 0b0 | If EDECCR.SE2 is 0, then Exception Catch debug events are disabled for Secure EL2. If EDECCR.SE2 is 1, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Secure EL2. |
| 0b1 | If EDECCR.SE2 is 0, then Exception Catch debug events are enabled for exception returns to Secure EL2. If EDECCR.SE2 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Secure EL2. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SR1, bit [9]

When FEAT_Debugv8p2 is implemented and Secure EL1 is implemented:

Controls exception catch on exception return to Secure EL1 in conjunction with EDECCR.SE1.

| SR1 | Meaning |
|------------|--|
| 0b0 | If EDECCR.SE1 is 0, then Exception Catch debug events are disabled for Secure EL1. If EDECCR.SE1 is 1, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Secure EL1. |
| 0b1 | If EDECCR.SE1 is 0, then Exception Catch debug events are enabled for exception returns to Secure EL1. If EDECCR.SE1 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Secure EL1. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SR0, bit [8]

When FEAT_Debugv8p2 is implemented and Secure EL0 is implemented:

Controls exception catch on exception return to Secure EL0.

| SR0 | Meaning |
|-----|---|
| 0b0 | Exception Catch debug events are disabled for Secure EL0. |
| 0b1 | Exception Catch debug events are enabled for exception returns to Secure EL0. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

NSE3, bit [7]

Access to this field is **RES0**.

NSE2, bit [6]

When FEAT_Debugv8p2 is implemented and Non-secure EL2 is implemented:

Controls exception catch on exception entry to Non-secure EL2. Also controls exception catch on exception return to Non-secure EL2 in conjunction with EDECCR.NSR2.

| NSE2 | Meaning |
|------|---|
| 0b0 | If EDECCR.NSR2 is 0, then Exception Catch debug events are disabled for Non-secure EL2. If EDECCR.NSR2 is 1, then Exception Catch debug events are enabled for exception returns to Non-secure EL2. |
| 0b1 | If EDECCR.NSR2 is 0, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Non-secure EL2. If EDECCR.NSR2 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Non-secure EL2. |

Note

It is IMPLEMENTATION DEFINED whether a reset entry to an Exception level will generate an Exception Catch debug event.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

When Non-secure EL2 is implemented:

Coarse-grained exception catch for Non-secure EL2. Controls Exception Catch debug events for Non-secure EL2.

| NSE2 | Meaning |
|-------------|---|
| 0b0 | Exception Catch debug events are disabled for Non-secure EL2. |
| 0b1 | Exception Catch debug events are enabled for Non-secure EL2. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

NSE1, bit [5]

When FEAT_Debugv8p2 is implemented and Non-secure EL1 is implemented:

Controls exception catch on exception entry to Non-secure EL1. Also controls exception catch on exception return to Non-secure EL1 in conjunction with EDECCR.NSR1.

| NSE1 | Meaning |
|-------------|---|
| 0b0 | If EDECCR.NSR1 is 0, then Exception Catch debug events are disabled for Non-secure EL1. If EDECCR.NSR1 is 1, then Exception Catch debug events are enabled for exception returns to Non-secure EL1. |
| 0b1 | If EDECCR.NSR1 is 0, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Non-secure EL1. If EDECCR.NSR1 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Non-secure EL1. |

Note

It is IMPLEMENTATION DEFINED whether a reset entry to an Exception level will generate an Exception Catch debug event.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

When Non-secure EL1 is implemented:

Coarse-grained exception catch for Non-secure EL1. Controls Exception Catch debug events for Non-secure EL1.

| NSE1 | Meaning |
|-------------|---|
| 0b0 | Exception Catch debug events are disabled for Non-secure EL1. |
| 0b1 | Exception Catch debug events are enabled for Non-secure EL1. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

NSE0, bit [4]

Access to this field is **RES0**.

SE3, bit [3]

When FEAT_Debugv8p2 is implemented and EL3 is implemented:

Controls exception catch on exception entry to EL3. Also controls exception catch on exception return to EL3 in conjunction with EDECCR.SR3.

| SE3 | Meaning |
|-----|---|
| 0b0 | If EDECCR.SR3 is 0, then Exception Catch debug events are disabled for EL3. If EDECCR.SR3 is 1, then Exception Catch debug events are enabled for exception returns to EL3. |
| 0b1 | If EDECCR.SR3 is 0, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to EL3. If EDECCR.SR3 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to EL3. |

Note

It is IMPLEMENTATION DEFINED whether a reset entry to an Exception level will generate an Exception Catch debug event.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

When FEAT_Debugv8p2 is not implemented and EL3 is implemented:

Coarse-grained exception catch for EL3. Controls Exception Catch debug events for EL3.

| SE3 | Meaning |
|-----|--|
| 0b0 | Exception Catch debug events are disabled for EL3. |
| 0b1 | Exception Catch debug events are enabled for EL3. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SE2, bit [2]

When FEAT_Debugv8p2 is implemented and FEAT_SEL2 is implemented:

Controls exception catch on exception entry to Secure EL2. Also controls exception catch on exception return to Secure EL2 in conjunction with EDECCR.SR2.

| SE2 | Meaning |
|-----|---|
| 0b0 | If EDECCR.SR2 is 0, then Exception Catch debug events are disabled for Secure EL2. If EDECCR.SR2 is 1, then Exception Catch debug events are enabled for exception returns to Secure EL2. |
| 0b1 | If EDECCR.SR2 is 0, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Secure EL2. If EDECCR.SR2 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Secure EL2. |

Note

It is IMPLEMENTATION DEFINED whether a reset entry to an Exception level will generate an Exception Catch debug event.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SE1, bit [1]

When FEAT_Debugv8p2 is implemented and Secure EL1 is implemented:

Controls exception catch on exception entry to Secure EL1. Also controls exception catch on exception return to Secure EL1 in conjunction with EDECCR.SR1.

| SE1 | Meaning |
|-----|---|
| 0b0 | If EDECCR.SR1 is 0, then Exception Catch debug events are disabled for Secure EL1. If EDECCR.SR1 is 1, then Exception Catch debug events are enabled for exception returns to Secure EL1. |
| 0b1 | If EDECCR.SR1 is 0, then Exception Catch debug events are enabled for exception entry, reset entry, and exception return to Secure EL1. If EDECCR.SR1 is 1, then Exception Catch debug events are enabled for exception entry and reset entry to Secure EL1. |

Note

It is IMPLEMENTATION DEFINED whether a reset entry to an Exception level will generate an Exception Catch debug event.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

When Secure EL1 is implemented:

Coarse-grained exception catch for Secure EL1. Controls Exception Catch debug events for Secure EL1.

| SE1 | Meaning |
|-----|---|
| 0b0 | Exception Catch debug events are disabled for Secure EL1. |
| 0b1 | Exception Catch debug events are enabled for Secure EL1. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SE0, bit [0]

Access to this field is **RES0**.

Accessing EDECCR

EDECCR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0x098 | EDECCR |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDECR, External Debug Execution Control Register

The EDECR characteristics are:

Purpose

Controls Halting debug events.

Configuration

If FEAT_DoPD is implemented, this register is in the Core power domain.

If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

EDECR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|----|-----|-------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | SS | RCE | OSUCE |

Bits [31:3]

Reserved, RES0.

SS, bit [2]

Halting step enable. Possible values of this field are:

| SS | Meaning |
|-----|------------------------------------|
| 0b0 | Halting step debug event disabled. |
| 0b1 | Halting step debug event enabled. |

If the value of EDECR.SS is changed when the PE is in Non-debug state, behavior is CONSTRAINED UNPREDICTABLE as described in 'Changing the value of EDECR.SS when not in Debug state'.

The reset behavior of this field is:

- On a Cold reset, when FEAT_DoPD is implemented, this field resets to 0.
- On an External debug reset, when FEAT_DoPD is not implemented, this field resets to 0.

RCE, bit [1]

When FEAT_DoPD is not implemented:

Reset Catch Enable.

| RCE | Meaning |
|-----|-----------------------------------|
| 0b0 | Reset Catch debug event disabled. |
| 0b1 | Reset Catch debug event enabled. |

The reset behavior of this field is:

- On an External debug reset, this field resets to 0.

Otherwise:

Reserved, RES0.

OSUCE, bit [0]**When FEAT_DoPD is not implemented:**

OS Unlock Catch Enable.

| OSUCE | Meaning |
|-------|---------------------------------------|
| 0b0 | OS Unlock Catch debug event disabled. |
| 0b1 | OS Unlock Catch debug event enabled. |

The reset behavior of this field is:

- On an External debug reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Accessing EDECR

EDECR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0x024 | EDECR |

This interface is accessible as follows:

- When (FEAT_DoPD is not implemented or IsCorePowered()) and SoftwareLockStatus() accesses to this register are **RO**.
- When (FEAT_DoPD is not implemented or IsCorePowered()) and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDES, External Debug Event Status Register

The EDES characteristics are:

Purpose

Indicates the status of internally pending Halting debug events.

Configuration

EDES is in the Core power domain.

Attributes

EDES is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|----|----|----|------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | EC | SS | RC | OSUC |

Bits [31:4]

Reserved, RES0.

EC, bit [3]

When FEAT_Debugv8p8 is implemented:

Exception Catch debug event pending.

| EC | Meaning |
|-----|---|
| 0b0 | Exception Catch debug event is not pending. |
| 0b1 | Exception Catch debug event is pending. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

SS, bit [2]

When FEAT_DoPD is implemented:

Halting step debug event pending. Possible values of this field are:

| SS | Meaning |
|-----|--|
| 0b0 | Reading this means that a Halting step debug event is not pending. Writing this means no action. |
| 0b1 | Reading this means that a Halting step debug event is pending. Writing this clears the pending Halting step debug event. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Halting step debug event pending. Possible values of this field are:

| SS | Meaning |
|-----|--|
| 0b0 | Reading this means that a Halting step debug event is not pending. Writing this means no action. |
| 0b1 | Reading this means that a Halting step debug event is pending. Writing this clears the pending Halting step debug event. |

The reset behavior of this field is:

- On a Warm reset, this field resets to the value in [EDEC.R](#).SS.

RC, bit [1]

Reset Catch debug event pending. Possible values of this field are:

| RC | Meaning |
|-----|--|
| 0b0 | Reading this means that a Reset Catch debug event is not pending. Writing this means no action. |
| 0b1 | Reading this means that a Reset Catch debug event is pending. Writing this clears the pending Reset Catch debug event. |

The reset behavior of this field is:

- On a Warm reset:
 - When FEAT_DoPD is implemented, this field resets to the value in [CTIDEVCTL](#).RCE.
 - When FEAT_DoPD is not implemented, this field resets to the value in [EDEC.R](#).RCE.

OSUC, bit [0]

OS Unlock Catch debug event pending. Possible values of this field are:

| OSUC | Meaning |
|------|---|
| 0b0 | Reading this means that an OS Unlock Catch debug event is not pending. Writing this means no action. |
| 0b1 | Reading this means that an OS Unlock Catch debug event is pending. Writing this clears the pending OS Unlock Catch debug event. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing EDES

If a request to clear a pending Halting debug event is received at or about the time when halting becomes allowed, it is **CONSTRAINED UNPREDICTABLE** whether the event is taken.

If Core power is removed while a Halting debug event is pending, it is lost. However, it might become pending again when the Core is powered back on and Cold reset.

EDES can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0x020 | EDES |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.

- When IsCorePowered(), !DoubleLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDITCTRL, External Debug Integration mode Control register

The EDITCTRL characteristics are:

Purpose

Enables the external debug to switch from its default mode into integration mode, where test software can control directly the inputs and outputs of the PE, for integration testing or topology detection.

Configuration

It is IMPLEMENTATION DEFINED whether EDITCTRL is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is OPTIONAL.

Attributes

EDITCTRL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | IME | | | | | | | | | | | | | | | |

Bits [31:1]

Reserved, RES0.

IME, bit [0]

Integration mode enable. When IME == 1, the device reverts to an integration mode to enable integration testing or topology detection. The integration mode behavior is IMPLEMENTATION DEFINED.

| IME | Meaning |
|-----|---------------------------|
| 0b0 | Normal operation. |
| 0b1 | Integration mode enabled. |

The following resets apply:

- Whichever power domain the register is implemented in, this field resets to 0.
- Otherwise, the value of this field is unchanged.

Accessing EDITCTRL

EDITCTRL can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xF00 | EDITCTRL |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register are **IMPDEF**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDITR, External Debug Instruction Transfer Register

The EDITR characteristics are:

Purpose

Used in Debug state for passing instructions to the PE for execution.

Configuration

EDITR is in the Core power domain.

Attributes

EDITR is a 32-bit register.

Field descriptions

When AArch32 is supported and in AArch32 state:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| hw2 | | | | | | | | | | | | | | | | hw1 | | | | | | | | | | | | | | | |

hw2, bits [31:16]

Second halfword of the T32 instruction to be executed on the PE. When EDITR contains a 16-bit T32 instruction, this field is ignored. For more information, see 'Behavior in Debug state'.

Note

The hw2 field is displayed on the left. This is not the usual convention for display of T32 instruction halfwords.

hw1, bits [15:0]

First halfword of the T32 instruction to be executed on the PE.

Note

The hw1 field is displayed on the right. This is not the usual convention for display of T32 instruction halfwords.

When AArch64 is supported and in AArch64 state:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| A64 instruction to be executed on the PE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

A64 instruction to be executed on the PE.

Accessing EDITR

If `EDSCR.ITE == 0` when the PE exits Debug state on receiving a Restart request trigger event, the behavior of any instruction issued through the ITR in Normal access mode that has not completed execution is **CONSTRAINED UNPREDICTABLE**, and must do one of the following:

- It must complete execution in Debug state before the PE executes the restart sequence.
- It must complete execution in Non-debug state before the PE executes the restart sequence.
- It must be abandoned. This means that the instruction does not execute. Any registers or memory accessed by the instruction are left in an UNKNOWN state.

EDITR ignores writes if the PE is in Non-debug state.

EDITR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0x084 | EDITR |

This interface is accessible as follows:

- When `IsCorePowered()`, `!DoubleLockStatus()`, `!OSLockStatus()` and `SoftwareLockStatus()` accesses to this register are **WI**.
- When `IsCorePowered()`, `!DoubleLockStatus()`, `!OSLockStatus()` and `!SoftwareLockStatus()` accesses to this register are **WO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDLAR, External Debug Lock Access Register

The EDLAR characteristics are:

Purpose

Allows or disallows access to the external debug registers through a memory-mapped interface.

The optional Software Lock provides a lock to prevent memory-mapped writes to the debug registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the debug registers. It does not, and cannot, prevent all accidental or malicious damage.

Configuration

If FEAT_DoPD is implemented, Software Lock is not implemented by the architecturally-defined debug components of the PE in the Core power domain.

If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Software uses EDLAR to set or clear the lock, and [EDLSR](#) to check the current status of the lock.

Attributes

EDLAR is a 32-bit register.

Field descriptions

When Software Lock is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | KEY | | | | | | | | | | | | | | | |

KEY, bits [31:0]

Lock Access control. Writing the key value 0xC5ACCE55 to this field unlocks the lock, enabling write accesses to this component's registers through a memory-mapped interface.

Writing any other value to this register locks the lock, disabling write accesses to this component's registers through a memory mapped interface.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |

Otherwise

Bits [31:0]

Reserved, RES0.

Accessing EDLAR

EDLAR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xFB0 | EDLAR |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **WO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDLSR, External Debug Lock Status Register

The EDLSR characteristics are:

Purpose

Indicates the current status of the software lock for external debug registers.

The optional Software Lock provides a lock to prevent memory-mapped writes to the debug registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the debug registers. It does not, and cannot, prevent all accidental or malicious damage.

Configuration

If FEAT_DoPD is implemented, Software Lock is not implemented by the architecturally-defined debug components of the PE in the Core power domain.

If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Software uses [EDLAR](#) to set or clear the lock, and EDLSR to check the current status of the lock.

Attributes

EDLSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|-----|-----|-----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | nTT | SLK | SLI |

Bits [31:3]

Reserved, RES0.

nTT, bit [2]

Not thirty-two bit access required. RAZ.

SLK, bit [1]

When Software Lock is implemented:

Software Lock status for this component. For an access to LSR that is not a memory-mapped access, or when Software Lock is not implemented, this field is RES0.

For memory-mapped accesses when Software Lock is implemented, possible values of this field are:

| SLK | Meaning |
|-----|---|
| 0b0 | Lock clear. Writes are permitted to this component's registers. |
| 0b1 | Lock set. Writes to this component's registers are ignored, and reads have no side effects. |

The reset behavior of this field is:

- On a Cold reset, when FEAT_DoPD is implemented, this field resets to 1.
- On an External debug reset, when FEAT_DoPD is not implemented, this field resets to 1.

Otherwise:

Reserved, RAZ.

SLI, bit [0]

Software Lock implemented. For an access to LSR that is not a memory-mapped access, this field is RAZ. For memory-mapped accesses, the value of this field is IMPLEMENTATION DEFINED. Permitted values are:

| SLI | Meaning |
|-----|--|
| 0b0 | Software Lock not implemented or not memory-mapped access. |
| 0b1 | Software Lock implemented and memory-mapped access. |

Accessing EDLSR**EDLSR can be accessed through the memory-mapped interfaces:**

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xFB4 | EDLSR |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPCSR, External Debug Program Counter Sample Register

The EDPCSR characteristics are:

Purpose

Holds a sampled instruction address value.

Configuration

EDPCSR is in the Core power domain.

This register is present only when FEAT_PCSRv8 is implemented and FEAT_PCSRv8p2 is not implemented. Otherwise, direct accesses to EDPCSR are RES0.

EDPCSR[63:32] and EDPCSR[31:0] are accessed at 32-bit memory mapped addresses that are not contiguous.

If FEAT_VHE is implemented, the format of this register differs depending on the value of [EDSCR.SC2](#).

Implemented only if the OPTIONAL PC Sample-based Profiling Extension is implemented in the external debug registers space.

Note

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors registers space.

Attributes

EDPCSR is a 64-bit register.

Field descriptions

When FEAT_VHE is not implemented or EDSCR.SC2 == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| PC Sample high word, EDPCSRhi | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PC Sample low word | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

PC Sample high word, EDPCSRhi. If [EDVIDSR.HV](#) == 0 then this field is RAZ, otherwise bits [63:32] of the sampled instruction address value. The translation regime that EDPCSR samples can be determined from [EDVIDSR.{NS,E2,E3}](#).

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

PC Sample low word. EDPCSRlo, bits[31:0] of the sampled instruction address value.

EDPCSRlo reads as 0xFFFFFFFF when any of the following are true:

- The PE is in Debug state.
- PC Sample-based profiling is prohibited.

If a branch instruction has retired since the PE left reset state, then the first read of EDPCSR[31:0] is permitted but not required to return 0xFFFFFFFF.

EDPCSRlo reads as an UNKNOWN value when any of the following are true:

- The PE is in reset state.
- No branch instruction has retired since the PE left reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
- No branch instruction has retired since the last read of EDPCSR[31:0].

For the cases where a read of EDPCSR[31:0] returns 0xFFFFFFFF or an UNKNOWN value, the read has the side-effect of setting EDPCSRhi, [EDCIDSR](#), and [EDVIDSR](#) to UNKNOWN values.

Otherwise, a read of EDPCSR[31:0] returns bits [31:0] of the sampled instruction address value and has the side-effect of indirectly writing to EDPCSRhi, [EDCIDSR](#), and [EDVIDSR](#). The translation regime that EDPCSR samples can be determined from [EDVIDSR](#).{NS,E2,E3}.

For a read of EDPCSR[31:0] from the memory-mapped interface, if EDLSR.SLK == 1, meaning the OPTIONAL Software Lock is locked, then the side-effect of the access does not occur and EDPCSRhi, [EDCIDSR](#), and [EDVIDSR](#) are unchanged.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When FEAT_VHE is implemented and EDSCR.SC2 == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------|----|------|----|----|----|----|----|-------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| NS | EL | RES0 | | | | | | PC Sample high word, EDPCSRhi | | | | | | | | | | | | | | | | | | | | | | | | |
| PC Sample low word | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

NS, bit [63]

Non-secure state sample. Indicates the Security state that is associated with the most recent EDPCSR sample or, when it is read as a single atomic 64-bit read, the current EDPCSR sample. The translation regime that EDPCSR samples can be determined from EDPCSR.{NS,EL}.

If EL3 is not implemented, this bit indicates the Effective value of SCR.NS.

| NS | Meaning |
|-----|----------------------------------|
| 0b0 | Sample is from Secure state. |
| 0b1 | Sample is from Non-secure state. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

EL, bits [62:61]

Exception level status sample. Indicates the Exception level that is associated with the most recent EDPCSR sample or, when it is read as a single atomic 64-bit read, the current EDPCSR sample. The translation regime that EDPCSR samples can be determined from EDPCSR.{NS,EL}.

| EL | Meaning |
|------|---------------------|
| 0b00 | Sample is from EL0. |
| 0b01 | Sample is from EL1. |
| 0b10 | Sample is from EL2. |
| 0b11 | Sample is from EL3. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [60:56]

Reserved, RES0.

Bits [55:32]

PC Sample high word, EDPCSRhi. Bits [55:32] of the sampled instruction address value. The translation regime that EDPCSR samples can be determined from EDPCSR.{NS,EL}.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [31:0]

PC Sample low word. EDPCSRlo, bits[31:0] of the sampled instruction address value.

EDPCSRlo reads as 0xFFFFFFFF when any of the following are true:

- The PE is in Debug state.
- PC Sample-based profiling is prohibited.

If a branch instruction has retired since the PE left reset state, then the first read of EDPCSR[31:0] is permitted but not required to return 0xFFFFFFFF.

EDPCSRlo reads as an UNKNOWN value when any of the following are true:

- The PE is in reset state.
- No branch instruction has retired since the PE left reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
- No branch instruction has retired since the last read of EDPCSR[31:0].

For the cases where a read of EDPCSR[31:0] returns 0xFFFFFFFF or an UNKNOWN value, the read has the side-effect of setting EDPCSRhi, [EDCIDSR](#), and [EDVIDSR](#) to UNKNOWN values.

Otherwise, a read of EDPCSR[31:0] returns bits [31:0] of the sampled instruction address value and has the side-effect of indirectly writing to EDPCSRhi, [EDCIDSR](#), and [EDVIDSR](#). The translation regime that EDPCSR samples can be determined from EDPCSR.{NS,EL}.

For a read of EDPCSR[31:0] from the memory-mapped interface, if EDLSR.SLK == 1, meaning the OPTIONAL Software Lock is locked, then the side-effect of the access does not occur and EDPCSRhi, [EDCIDSR](#), and [EDVIDSR](#) are unchanged.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing EDPCSR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'

EDPCSR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance | Range |
|-----------|--------|----------|-------|
| Debug | 0x0A0 | EDPCSR | 31:0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

| Component | Offset | Instance | Range |
|-----------|--------|----------|-------|
| Debug | 0x0AC | EDPCSR | 63:32 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPFR, External Debug Processor Feature Register

The EDPFR characteristics are:

Purpose

Provides information about implemented PE features.

For general information about the interpretation of the ID registers, see 'Principles of the ID scheme for fields in ID registers'.

Configuration

It is IMPLEMENTATION DEFINED whether EDPFR is implemented in the Core power domain or in the Debug power domain.

Attributes

EDPFR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN |
| UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN | UNKNOWN |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:60]

From Armv8.5:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

Bits [59:56]

From Armv8.5:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

Bits [55:52]

Reserved, RES0.

Bits [51:48]

From Armv8.4:

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

AMU, bits [47:44]

Indicates support for Activity Monitors Extension. Defined values are:

| AMU | Meaning |
|--------|--|
| 0b0000 | Activity Monitors Extension is not implemented. |
| 0b0001 | FEAT_AMUv1 is implemented. |
| 0b0010 | FEAT_AMUv1p1 is implemented. As 0b0001 and adds support for virtualization of the activity monitor event counters. |

All other values are reserved.

FEAT_AMUv1 implements the functionality identified by the value 0b0001.

FEAT_AMUv1p1 implements the functionality identified by the value 0b0010.

In Armv8.0, the only permitted value is 0b0000.

In Armv8.4, the permitted values are 0b0000 and 0b0001.

From Armv8.6, the permitted values are 0b0000, 0b0001, and 0b0010.

Bits [43:40]**From Armv8.2:**

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

SEL2, bits [39:36]

Secure EL2. Defined values are:

| SEL2 | Meaning |
|--------|--------------------------------|
| 0b0000 | Secure EL2 is not implemented. |
| 0b0001 | Secure EL2 is implemented. |

All other values are reserved.

SVE, bits [35:32]

Scalable Vector Extension. Defined values are:

| SVE | Meaning |
|--------|-------------------------|
| 0b0000 | SVE is not implemented. |
| 0b0001 | SVE is implemented. |

All other values are reserved.

Bits [31:28]**From Armv8.2:**

Reserved, UNKNOWN.

Otherwise:

Reserved, RES0.

GIC, bits [27:24]

System register GIC interface support. Defined values are:

| GIC | Meaning |
|------------|--|
| 0b0000 | GIC CPU interface system registers not implemented. |
| 0b0001 | System register interface to versions 3.0 and 4.0 of the GIC CPU interface is supported. |
| 0b0011 | System register interface to version 4.1 of the GIC CPU interface is supported. |

All other values are reserved.

In an Armv8-A implementation that supports AArch64, this field returns the value of [ID_AA64PFR0_EL1](#).GIC.

AdvSIMD, bits [23:20]

Advanced SIMD. Defined values are:

| AdvSIMD | Meaning |
|----------------|--|
| 0b0000 | Advanced SIMD is implemented, including support for the following Sisd and SIMD operations: <ul style="list-style-type: none"> Integer byte, halfword, word and doubleword element operations. Single-precision and double-precision floating-point arithmetic. Conversions between single-precision and half-precision data types, and double-precision and half-precision data types. |
| 0b0001 | As for 0b0000, and also includes support for half-precision floating-point arithmetic. |
| 0b1111 | Advanced SIMD is not implemented. |

All other values are reserved.

This field must have the same value as the FP field.

The permitted values are:

- 0b0000 in an implementation with Advanced SIMD support, that does not include the FEAT_FP16 extension.
- 0b0001 in an implementation with Advanced SIMD support, that includes the FEAT_FP16 extension.
- 0b1111 in an implementation without Advanced SIMD support.

In an Armv8-A implementation that supports AArch64, this field returns the value of [ID_AA64PFR0_EL1](#).AdvSIMD.

FP, bits [19:16]

Floating-point. Defined values are:

| FP | Meaning |
|-----------|---|
| 0b0000 | Floating-point is implemented, and includes support for: <ul style="list-style-type: none"> Single-precision and double-precision floating-point types. Conversions between single-precision and half-precision data types, and double-precision and half-precision data types. |
| 0b0001 | As for 0b0000, and also includes support for half-precision floating-point arithmetic. |
| 0b1111 | Floating-point is not implemented. |

All other values are reserved.

This field must have the same value as the AdvSIMD field.

The permitted values are:

- 0b0000 in an implementation with floating-point support, that does not include the FEAT_FP16 extension.
- 0b0001 in an implementation with floating-point support, that includes the FEAT_FP16 extension.
- 0b1111 in an implementation without floating-point support.

In an Armv8-A implementation that supports AArch64, this field returns the value of [ID_AA64PFR0_EL1](#).FP.

EL3, bits [15:12]

AArch64 EL3 Exception level handling. Defined values are:

| EL3 | Meaning |
|--------|--|
| 0b0000 | EL3 is not implemented or cannot be executed in AArch64 state. |
| 0b0001 | EL3 can be executed in AArch64 state only. |
| 0b0010 | EL3 can be executed in both Execution states. |

When the value of [EDAA32PFR](#).EL3 is non-zero, this field must be 0b0000.

All other values are reserved.

In an Armv8-A implementation that supports AArch64, this field returns the value of [ID_AA64PFR0_EL1](#).EL3.

EL2, bits [11:8]

AArch64 EL2 Exception level handling. Defined values are:

| EL2 | Meaning |
|--------|--|
| 0b0000 | EL2 is not implemented or cannot be executed in AArch64 state. |
| 0b0001 | EL2 can be executed in AArch64 state only. |
| 0b0010 | EL2 can be executed in both Execution states. |

When the value of [EDAA32PFR](#).EL2 is non-zero, this field must be 0b0000.

All other values are reserved.

In an Armv8-A implementation that supports AArch64, this field returns the value of [ID_AA64PFR0_EL1](#).EL2.

EL1, bits [7:4]

AArch64 EL1 Exception level handling. Defined values are:

| EL1 | Meaning |
|--------|---|
| 0b0000 | EL1 cannot be executed in AArch64 state. |
| | EL1 can be executed in AArch32 state only. |
| 0b0001 | EL1 can be executed in AArch64 state only. |
| 0b0010 | EL1 can be executed in both Execution states. |

All other values are reserved.

In an Armv8-A implementation that supports AArch64, this field returns the value of [ID_AA64PFR0_EL1](#).EL1.

EL0, bits [3:0]

AArch64 EL0 Exception level handling. Defined values are:

| EL0 | Meaning |
|--------|---|
| 0b0000 | EL0 cannot be executed in AArch64 state. |
| | EL0 can be executed in AArch32 state only. |
| 0b0001 | EL0 can be executed in AArch64 state only. |
| 0b0010 | EL0 can be executed in both Execution states. |

All other values are reserved.

In an Armv8-A implementation that supports AArch64, this field returns the value of [ID_AA64PFR0_EL1](#).EL0.

Accessing EDPFR

EDPFR can be accessed through the external debug interface:

| Component | Offset | Instance | Range |
|-----------|--------|----------|-------|
| Debug | 0xD20 | EDPFR | 31:0 |

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register are **IMPDEF**.

| Component | Offset | Instance | Range |
|-----------|--------|----------|-------|
| Debug | 0xD24 | EDPFR | 63:32 |

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register are **IMPDEF**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPIDR0, External Debug Peripheral Identification Register 0

The EDPIDR0 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

EDPIDR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|--------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PART_0 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, least significant byte.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDPIDR0

EDPIDR0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xFE0 | EDPIDR0 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

EDPIDR1, External Debug Peripheral Identification Register 1

The EDPIDR1 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

EDPIDR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---|--------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | DES_0 | | | | PART_1 | | | |

Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, least significant nibble of JEP106 ID code. For Arm Limited, this field is 0b1011.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PART_1, bits [3:0]

Part number, most significant nibble.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDPIDR1

EDPIDR1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xFE4 | EDPIDR1 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPIDR2, External Debug Peripheral Identification Register 2

The EDPIDR2 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

EDPIDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|-------|----|-------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | REVISION | | | | JEDEC | | DES_1 | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Part major revision. Parts can also use this field to extend Part number to 16-bits.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

Indicates a JEP106 identity code is used.

Reads as 0b1.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, most significant bits of JEP106 ID code. For Arm Limited, this field is 0b011.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDPIDR2

EDPIDR2 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xFE8 | EDPIDR2 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPIDR3, External Debug Peripheral Identification Register 3

The EDPIDR3 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

EDPIDR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|--------|---|---|---|------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | REVAND | | | | CMOD | | | |

Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Part minor revision. Parts using [EDPIDR2](#).REVISION as an extension to the Part number must use this field as a major revision number.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer modified. Indicates someone other than the Designer has modified the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDPIDR3

EDPIDR3 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
|-----------|--------|----------|

| | | |
|-------|-------|---------|
| Debug | 0xFEC | EDPIDR3 |
|-------|-------|---------|

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPIDR4, External Debug Peripheral Identification Register 4

The EDPIDR4 characteristics are:

Purpose

Provides information to identify an external debug component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

EDPIDR4 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|------|---|---|-------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | SIZE | | | DES_2 | | | | |

Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

Size of the component. Log₂ of the number of 4KB pages from the start of the component to the end of the component ID registers.

Reads as 0b0000.

Access to this field is **RO**.

DES_2, bits [3:0]

Designer, JEP106 continuation code, least significant nibble. For Arm Limited, this field is 0b0100.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing EDPIDR4

EDPIDR4 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
|-----------|--------|----------|

| | | |
|-------|-------|---------|
| Debug | 0xFD0 | EDPIDR4 |
|-------|-------|---------|

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPRCR, External Debug Power/Reset Control Register

The EDPRCR characteristics are:

Purpose

Controls the PE functionality related to powerup, reset, and powerdown.

Configuration

EDPRCR contains fields that are in the Core power domain and fields that are in the Debug power domain.

If FEAT_DoPD is implemented then all fields in this register are in the Core power domain.

CORENPDRQ is the only field that is mapped between the EDPRCR and DBGPRCR and DBGPRCR_EL1.

Attributes

EDPRCR is a 32-bit register.

Field descriptions

When FEAT_DoPD is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|------|-----------|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | CWRR | CORENPDRQ | | |

Bits [31:2]

Reserved, RES0.

CWRR, bit [1]

When FEAT_RME is implemented:

The PE ignores all writes to this bit.

Otherwise:

Warm reset request.

The extent of the reset is IMPLEMENTATION DEFINED, but must be one of:

- The request is ignored.
- Only this PE is Warm reset.
- This PE and other components of the system, possibly including other PEs, are Warm reset.

Arm deprecates use of this bit, and recommends that implementations ignore the request.

| CWRR | Meaning |
|------|---------------------|
| 0b0 | No action. |
| 0b1 | Request Warm reset. |

This field is in the Core power domain

The PE ignores writes to this bit if any of the following are true:

- ExternalInvasiveDebugEnabled() == FALSE, EL3 is not implemented, and the implemented Security state is Non-secure state.
- ExternalSecureInvasiveDebugEnabled() == FALSE, EL3 is not implemented, and the implemented Security state is Secure state.
- ExternalSecureInvasiveDebugEnabled() == FALSE and EL3 is implemented.

In an implementation that includes the recommended external debug interface, this bit drives the DBGRSTREQ signal.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing this field has the following behavior:

- **RAZ/WI** if any of the following are true:
 - OSLockStatus()
 - SoftwareLockStatus()
- Otherwise, access to this field is **WO/RAZ**.

CORENPDRQ, bit [0]

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

| CORENPDRQ | Meaning |
|-----------|--|
| 0b0 | If the system responds to a powerdown request, it powers down Core power domain. |
| 0b1 | If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain. |

When this bit reads as UNKNOWN, the PE ignores writes to this bit.

This field is in the Core power domain, and permitted accesses to this field map to the [DBGPRCR.CORENPDRQ](#) and [DBGPRCR_EL1.CORENPDRQ](#) fields.

In an implementation that includes the recommended external debug interface, this bit drives the DBGNOPWRDWN signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to the Cold reset value on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states, see 'Core power domain power states'.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

On a Cold reset, if the powerup request is implemented and the powerup request has been asserted, this field is an IMPLEMENTATION DEFINED choice of 0 or 1. If the powerup request is not asserted, this field is set to 0.

Accessing this field has the following behavior:

- When OSLockStatus(), access to this field is **UNKNOWN/WI**.
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|------|----|------|----|-----------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | COREPURQ | | RES0 | | CWRR | | CORENPDRQ | | | | | | | | | |

Bits [31:4]

Reserved, RES0.

COREPURQ, bit [3]

Core powerup request. Allows a debugger to request that the power controller power up the core, enabling access to the debug register in the Core power domain, and that the power controller emulates powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

| COREPURQ | Meaning |
|----------|--|
| 0b0 | Do not request power up of the Core power domain. |
| 0b1 | Request power up of the Core power domain, and emulation of powerdown. |

In an implementation that includes the recommended external debug interface, this bit drives the DBGPWRUPREQ signal.

This field is in the Debug power domain and can be read and written when the Core power domain is powered off.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The reset behavior of this field is:

- On an External debug reset, this field resets to 0.

Accessing this field has the following behavior:

- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Bit [2]

Reserved, RES0.

CWRR, bit [1]

When FEAT_RME is implemented:

The PE ignores all writes to this bit.

Otherwise:

Warm reset request.

The extent of the reset is IMPLEMENTATION DEFINED, but must be one of:

- The request is ignored.
- Only this PE is Warm reset.
- This PE and other components of the system, possibly including other PEs, are Warm reset.

Arm deprecates use of this bit, and recommends that implementations ignore the request.

| CWRR | Meaning |
|------|---------------------|
| 0b0 | No action. |
| 0b1 | Request Warm reset. |

This field is in the Core power domain

The PE ignores writes to this bit if any of the following are true:

- ExternalInvasiveDebugEnabled() == FALSE, EL3 is not implemented, and the implemented Security state is Non-secure state.
- ExternalSecureInvasiveDebugEnabled() == FALSE, EL3 is not implemented, and the implemented Security state is Secure state.
- ExternalSecureInvasiveDebugEnabled() == FALSE and EL3 is implemented.

In an implementation that includes the recommended external debug interface, this bit drives the DBGRSTREQ signal.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing this field has the following behavior:

- **RAZ/WI** if any of the following are true:
 - !IsCorePowered()
 - DoubleLockStatus()
 - OSLockStatus()
 - SoftwareLockStatus()
- Otherwise, access to this field is **WO/RAZ**.

CORENPDRQ, bit [0]

Core no powerdown request. Requests emulation of powerdown.

This request is typically passed to an external power controller. This means that whether a request causes power up is dependent on the IMPLEMENTATION DEFINED nature of the system. The power controller must not allow the Core power domain to switch off while this bit is 1.

| CORENPDRQ | Meaning |
|-----------|--|
| 0b0 | If the system responds to a powerdown request, it powers down Core power domain. |
| 0b1 | If the system responds to a powerdown request, it does not powerdown the Core power domain, but instead emulates a powerdown of that domain. |

When this bit reads as UNKNOWN, the PE ignores writes to this bit.

This field is in the Core power domain, and permitted accesses to this field map to the [DBGPRCR](#).CORENPDRQ and [DBGPRCR_EL1](#).CORENPDRQ fields.

In an implementation that includes the recommended external debug interface, this bit drives the DBGNOPWRDWN signal.

It is IMPLEMENTATION DEFINED whether this bit is reset to the value of [EDPRCR](#).COREPURQ on exit from an IMPLEMENTATION DEFINED software-visible retention state. For more information about retention states, see 'Core power domain power states'.

Note

Writes to this bit are not prohibited by the IMPLEMENTATION DEFINED authentication interface. This means that a debugger can request emulation of powerdown regardless of whether invasive debug is permitted.

The reset behavior of this field is:

- On a Cold reset, this field resets to the value in [EDPRCR](#).COREPURQ.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - !IsCorePowered()
 - DoubleLockStatus()
 - OSLockStatus()
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RW**.

Accessing EDPRCR

On permitted accesses to the register, other access controls affect the behavior of some fields. See the field descriptions for more information.

EDPRCR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0x310 | EDPRCR |

This interface is accessible as follows:

- When (FEAT_DoPD is not implemented or IsCorePowered()) and SoftwareLockStatus() accesses to this register are **RO**.
- When (FEAT_DoPD is not implemented or IsCorePowered()) and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDPRSR, External Debug Processor Status Register

The EDPRSR characteristics are:

Purpose

Holds information about the reset and powerdown state of the PE.

Configuration

EDPRSR contains fields that are in the Core power domain and fields that are in the Debug power domain.

If FEAT_DoPD is implemented then all fields in this register are in the Core power domain.

Attributes

EDPRSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|-------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|-------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|-----|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|-------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|-------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|-----|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--------|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|----|--|--|--|--|--|--|--|--|--|--|--|--|--|--|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | EPMAD | | | | | | | | | | | | | | | | ETADE | | | | | | | | | | | | | | | | EDADE | | | | | | | | | | | | | | | | STAD | | | | | | | | | | | | | | | | ETAD | | | | | | | | | | | | | | | | SDR | | | | | | | | | | | | | | | | SPMAD | | | | | | | | | | | | | | | | EPMAD | | | | | | | | | | | | | | | | SDAD | | | | | | | | | | | | | | | | EDAD | | | | | | | | | | | | | | | | DLK | | | | | | | | | | | | | | | | OSLK | | | | | | | | | | | | | | | | HALTED | | | | | | | | | | | | | | | | SP | | | | | | | | | | | | | | | |

Bits [31:17]

Reserved, RES0.

EPMAD, bit [16]

When FEAT_RME is implemented and FEAT_PMUv3 is implemented:

External Performance Monitors Access Disable Extended Status. Together with EDPRSR.EPMAD, reports whether access to Performance Monitor registers by an external debugger is permitted.

For a description of the values derived by evaluating EPMAD and EPMAD together, see EDPRSR.EPMAD.

Otherwise:

Reserved, RES0.

ETADE, bit [15]

When FEAT_RME is implemented, external debugger access to the PE Trace Unit registers is implemented and FEAT_TRBE is implemented:

External Trace Access Disable Extended Status. Together with EDPRSR.ETAD, reports whether access to PE Trace Unit registers by an external debugger is permitted.

For a description of the values derived by evaluating ETAD and ETAD together, see EDPRSR.ETAD.

Otherwise:

Reserved, RES0.

EDADE, bit [14]**When FEAT_RME is implemented:**

External Debug Access Disable Extended Status. Together with EDPRSR.EDAD, reports whether access to breakpoint registers, watchpoint registers, and [OSLAR_EL1](#) by an external debugger is permitted.

For a description of the values derived by evaluating EDAD and EDADE together, see EDPRSR.EDAD.

Otherwise:

Reserved, RES0.

STAD, bit [13]**When external debugger access to the PE Trace Unit registers is implemented and FEAT_TRBE is implemented:**

Sticky ETAD error. Set to 1 when a Non-secure external debug interface access to an external trace register returns an error because AllowExternalTraceAccess() == FALSE for the access.

| STAD | Meaning |
|------|---|
| 0b0 | Since EDPRSR was last read, no external accesses to the PE Trace Unit registers have failed because AllowExternalTraceAccess() was FALSE for the access. |
| 0b1 | Since EDPRSR was last read, at least one external access to the PE Trace Unit registers has failed because AllowExternalTraceAccess() was FALSE for the access. |

If IsCorePowered() == TRUE, the Core power domain is powered up, then, following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or DoubleLockStatus() == FALSE then this bit clears to 0.
- If FEAT_DoubleLock is implemented and DoubleLockStatus() == TRUE then it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This bit is in the Core power domain.

Note

If FEAT_DoPD is implemented, FEAT_DoubleLock is not implemented.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - DoubleLockStatus()
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - EDPRSR.R == 1
- Otherwise, access to this field is **RC/WI**.

Otherwise:

Reserved, RES0.

ETAD, bit [12]**When FEAT_RME is implemented, external debugger access to the PE Trace Unit registers is implemented and FEAT_TRBE is implemented:**

External Trace Access Disable Status. Together with EDPRSR.ETADE, reports whether access to PE Trace Unit registers by an external debugger is permitted.

| ETADE | ETAD | Meaning |
|-------|------|--|
| 0b0 | 0b0 | Access to PE Trace Unit registers by an external debugger is permitted. |
| 0b0 | 0b1 | Root and Secure access to PE Trace Unit registers by an external debugger is permitted. Realm and Non-secure access to PE Trace Unit registers by an external debugger is not permitted. |
| 0b1 | 0b0 | Root and Realm access to PE Trace Unit registers by an external debugger is permitted. Secure and Non-secure access to PE Trace Unit registers by an external debugger is not permitted. |
| 0b1 | 0b1 | Root access to PE Trace Unit registers by an external debugger is permitted. Secure, Non-secure, and Realm access to PE Trace Unit registers by an external debugger is not permitted. |

When external debugger access to the PE Trace Unit registers is implemented and FEAT_TRBE is implemented:

External Trace Access Disable status.

| ETAD | Meaning |
|------|--|
| 0b0 | External Non-secure PE Trace Unit accesses enabled. AllowExternalTraceAccess() == TRUE for a Non-secure access. |
| 0b1 | External Non-secure PE Trace Unit accesses disabled. AllowExternalTraceAccess() == FALSE for a Non-secure access. |

This bit is in the Core power domain.

Note

If FEAT_DoPD is implemented, FEAT_DoubleLock is not implemented.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - DoubleLockStatus()
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - EDPRSR.R == 1
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RES0.

SDR, bit [11]

Sticky Debug Restart. Set to 1 when the PE exits Debug state.

Permitted values are:

| SDR | Meaning |
|-----|--|
| 0b0 | The PE has not restarted since EDPRSR was last read. |
| 0b1 | The PE has restarted since EDPRSR was last read. |

Note

If a reset occurs when the PE is in Debug state, the PE exits Debug state. SDR is UNKNOWN on Warm reset, meaning a debugger must also use the SR bit to determine whether the PE has left Debug state.

If The Core power domain is powered up, then following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.
- If FEAT_DoubleLock is implemented and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - DoubleLockStatus()
 - EDPRSR.R == 1
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RC/WI**.

SPMAD, bit [10]

When FEAT_Debugv8p4 is implemented:

Sticky EPMAD error. Set to 1 if an external debug interface access to a Performance Monitors register returns an error because AllowExternalPMUAccess() == FALSE.

Permitted values are:

| SPMAD | Meaning |
|-------|---|
| 0b0 | No Non-secure external debug interface accesses to the external Performance Monitors registers have failed because AllowExternalPMUAccess() == FALSE for the access since EDPRSR was last read. |
| 0b1 | At least one Non-secure external debug interface access to the external Performance Monitors register has failed and returned an error because AllowExternalPMUAccess() == FALSE for the access since EDPRSR was last read. |

If the Core power domain is powered up, then, following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or DoubleLockStatus() == FALSE, this bit clears to 0.
- If FEAT_DoubleLock is implemented, and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - DoubleLockStatus()
 - EDPRSR.R == 1
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RC/WI**.

Otherwise:

Sticky EPMAD error.

| SPMAD | Meaning |
|-------|---|
| 0b0 | No external debug interface accesses to the Performance Monitors registers have failed because AllowExternalPMUAccess() == FALSE since EDPRSR was last read. |
| 0b1 | At least one external debug interface access to the Performance Monitors registers has failed and returned an error because AllowExternalPMUAccess() == FALSE since EDPRSR was last read. |

If the Core power domain is powered up, then, following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or DoubleLockStatus() == FALSE, this bit clears to 0.
- If FEAT_DoubleLock is implemented, and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - OSLockStatus()
 - DoubleLockStatus()
 - EDPRSR.R == 1
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RC/WI**.

EPMAD, bit [9]

When FEAT_RME is implemented and FEAT_PMUv3 is implemented:

External Performance Monitors Access Disable Status. Together with EDPRSR.EPMADE, reports whether access to Performance Monitor registers by an external debugger is permitted.

| EPMADE | EPMAD | Meaning |
|--------|-------|--|
| 0b0 | 0b0 | Access to Performance Monitor registers by an external debugger is permitted. |
| 0b0 | 0b1 | Root and Secure access to Performance Monitor registers by an external debugger is permitted. Realm and Non-secure access to Performance Monitor registers by an external debugger is not permitted. |
| 0b1 | 0b0 | Root and Realm access to Performance Monitor registers by an external debugger is permitted. Secure and Non-secure access to Performance Monitor registers by an external debugger is not permitted. |
| 0b1 | 0b1 | Root access to Performance Monitor registers by an external debugger is permitted. Secure, Non-secure, and Realm access to Performance Monitor registers by an external debugger is not permitted. |

When FEAT_Debugv8p4 is implemented and FEAT_PMUv3 is implemented:

External Performance Monitors Non-secure Access Disable status.

| EPMAD | Meaning |
|--------------|--|
| 0b0 | External Non-secure Performance Monitors access enabled. AllowExternalPMUAccess() == TRUE for a Non-secure access. |
| 0b1 | External Non-secure Performance Monitors access disabled. AllowExternalPMUAccess() == FALSE for a Non-secure access. |

This field is in the Core power domain.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - DoubleLockStatus()
 - EDPRSR.R == 1
- Otherwise, access to this field is **RO**.

When FEAT_PMUv3 is implemented:

External Performance Monitors access disable status.

| EPMAD | Meaning |
|--------------|---|
| 0b0 | External Performance Monitors access enabled. AllowExternalPMUAccess() == TRUE. |
| 0b1 | External Performance Monitors access disabled. AllowExternalPMUAccess() == FALSE. |

This field is in the Core power domain.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - OSLockStatus()
 - DoubleLockStatus()
 - EDPRSR.R == 1
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RES0.

SDAD, bit [8]

When FEAT_Debugv8p4 is implemented:

Sticky EDAD error. Set to 1 if an external debug interface access to a debug register returns an error because AllowExternalDebugAccess() == FALSE.

| SDAD | Meaning |
|-------------|--|
| 0b0 | No Non-secure external debug interface accesses to the debug registers have failed because AllowExternalDebugAccess() == FALSE for the access since EDPRSR was last read. |
| 0b1 | At least one Non-secure external debug interface access to the debug registers has failed and returned an error because AllowExternalDebugAccess() == FALSE for the access since EDPRSR was last read. |

If the Core power domain is powered up, then, following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.
- If FEAT_DoubleLock is implemented and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - DoubleLockStatus()
 - EDPRSR.R == 1
- Otherwise, access to this field is **RO**.

Otherwise:

Sticky EDAD error. Set to 1 if an external debug interface access to a debug register returns an error because AllowExternalDebugAccess() == FALSE.

| SDAD | Meaning |
|------|--|
| 0b0 | No external debug interface accesses to the debug registers have failed because AllowExternalDebugAccess() == FALSE since EDPRSR was last read. |
| 0b1 | At least one external debug interface access to the debug registers has failed and returned an error because AllowExternalDebugAccess() == FALSE since EDPRSR was last read. |

If the Core power domain is powered up, then, following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.
- If FEAT_DoubleLock is implemented and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This bit is UNKNOWN on reads if OSLockStatus() == TRUE and external debug writes to [OSLAR_EL1](#) do not return an error when AllowExternalDebugAccess() == FALSE.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - DoubleLockStatus()
 - EDPRSR.R == 1
- Otherwise, access to this field is **RO**.

EDAD, bit [7]

When FEAT_RME is implemented:

External Debug Access Disable Status. Together with EDPRSR.EDADE, reports whether access to breakpoint registers, watchpoint registers, and [OSLAR_EL1](#) by an external debugger is permitted.

| EDADE | EDAD | Meaning |
|-------|------|--|
| 0b0 | 0b0 | Access to Debug registers by an external debugger is permitted. |
| 0b0 | 0b1 | Root and Secure access to Debug registers by an external debugger is permitted. Realm and Non-secure access to Debug registers by an external debugger is not permitted. |
| 0b1 | 0b0 | Root and Realm access to Debug registers by an external debugger is permitted. Secure and Non-secure access to Debug registers by an external debugger is not permitted. |
| 0b1 | 0b1 | Root access to Debug registers by an external debugger is permitted. Secure, Non-secure, and Realm access to Debug registers by an external debugger is not permitted. |

When FEAT_Debugv8p4 is implemented:

External Debug Access Disable status.

| EDAD | Meaning |
|------|--|
| 0b0 | External Non-secure access to breakpoint registers, watchpoint registers, and OSLAR_EL1 enabled. AllowExternalDebugAccess() == TRUE for a Non-secure access. |
| 0b1 | External Non-secure access to breakpoint registers, watchpoint registers, and OSLAR_EL1 disabled. AllowExternalDebugAccess() == FALSE for a Non-secure access. |

This field is in the Core power domain.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - DoubleLockStatus()
 - EDPRSR.R == 1
- Otherwise, access to this field is **RO**.

When FEAT_Debugv8p2 is implemented:

External Debug Access Disable status.

| EDAD | Meaning |
|------|---|
| 0b0 | External access to breakpoint registers, watchpoint registers, and OSLAR_EL1 enabled. AllowExternalDebugAccess() == TRUE. |
| 0b1 | External access to breakpoint registers, watchpoint registers, and OSLAR_EL1 disabled. AllowExternalDebugAccess() == FALSE. |

This bit is not valid and reads UNKNOWN if OSLockStatus() == TRUE and external debug writes to [OSLAR_EL1](#) do not return an error when AllowExternalDebugAccess() == FALSE.

This field is in the Core power domain.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - DoubleLockStatus()
 - EDPRSR.R == 1
- Otherwise, access to this field is **RO**.

Otherwise:

External Debug Access Disable status.

| EDAD | Meaning |
|------|--|
| 0b0 | External access to breakpoint registers, watchpoint registers, and OSLAR_EL1 enabled. <code>AllowExternalDebugAccess() == TRUE</code> . |
| 0b1 | External access to breakpoint registers, watchpoint registers disabled. It is IMPLEMENTATION DEFINED whether accesses to OSLAR_EL1 are enabled or disabled. <code>AllowExternalDebugAccess() == FALSE</code> . |

This field is in the Core power domain.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - `(FEAT_DoPD is not implemented and !IsCorePowered())`
 - `DoubleLockStatus()`
 - `EDPRSR.R == 1`
- Otherwise, access to this field is **RO**.

DLK, bit [6]

When FEAT_Debugv8p4 is implemented:

This field is RES0.

When FEAT_Debugv8p2 is implemented and FEAT_DoubleLock is implemented:

Double Lock.

From Armv8.2, this field is deprecated.

This field is in the Core power domain.

Accessing this field has the following behavior:

- **RAZ/WI** if all of the following are true:
 - `IsCorePowered()`
 - `!DoubleLockStatus()`
- Otherwise, access to this field is **UNKNOWN/WI**.

When FEAT_DoubleLock is implemented:

Double Lock.

This field returns the result of the pseudocode function `DoubleLockStatus()`.

If the Core power domain is powered up and `DoubleLockStatus() == TRUE`, it is IMPLEMENTATION DEFINED whether:

- `EDPRSR.PU` reads as 1, `EDPRSR.DLK` reads as 1, and `EDPRSR.SPD` is UNKNOWN.
- `EDPRSR.PU` reads as 0, `EDPRSR.DLK` is UNKNOWN, and `EDPRSR.SPD` reads as 0.

This field is in the Core power domain.

| DLK | Meaning |
|-----|---|
| 0b0 | <code>DoubleLockStatus()</code> returns FALSE. |
| 0b1 | <code>DoubleLockStatus()</code> returns TRUE and the Core power domain is powered up. |

Accessing this field has the following behavior:

- **UNKNOWN/WI** if all of the following are true:
 - FEAT_DoPD is not implemented
 - !IsCorePowered()
- Otherwise, access to this field is **RO**.

Otherwise:

Reserved, RES0.

OSLK, bit [5]

OS Lock status bit.

A read of this bit returns the value of [OSLSR_EL1.OSLK](#).

This field is in the Core power domain.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if all of the following are true:
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - DoubleLockStatus()
 - EDPRSR.R == 1
- Otherwise, access to this field is **RO**.

HALTED, bit [4]

Halted status bit.

| HALTED | Meaning |
|--------|---------------------------|
| 0b0 | PE is in Non-debug state. |
| 0b1 | PE is in Debug state. |

This field is in the Core power domain.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if all of the following are true:
 - FEAT_DoPD is not implemented
 - !IsCorePowered()
- Otherwise, access to this field is **RO**.

SR, bit [3]

Sticky core Reset status bit.

Permitted values are:

| SR | Meaning |
|-----|---|
| 0b0 | The non-debug logic of the PE is not in reset state and has not been reset since the last time EDPRSR was read. |
| 0b1 | The non-debug logic of the PE is in reset state or has been reset since the last time EDPRSR was read. |

If EDPRSR.PU reads as 1 and EDPRSR.R reads as 0, which means that the Core power domain is in a powerup state and that the non-debug logic of the PE is not in reset state, then following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.
- If FEAT_DoubleLock is implemented and DoubleLockStatus() == TRUE, it is UNPREDICTABLE whether this bit clears to 0 or is unchanged.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - DoubleLockStatus()
- When SoftwareLockStatus(), access to this field is **RO**.
- Otherwise, access to this field is **RC/WI**.

R, bit [2]

PE Reset status bit.

Permitted values are:

| R | Meaning |
|-----|--|
| 0b0 | The non-debug logic of the PE is not in reset state. |
| 0b1 | The non-debug logic of the PE is in reset state. |

If FEAT_DoubleLock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a CONSTRAINED UNPREDICTABLE value. For more information, see 'EDPRSR.{DLK, R} and reset state'.

This field is in the Core power domain.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - (FEAT_DoPD is not implemented and !IsCorePowered())
 - DoubleLockStatus()
- Otherwise, access to this field is **RO**.

SPD, bit [1]

Sticky core Powerdown status bit.

If FEAT_DoubleLock is implemented and DoubleLockStatus() == TRUE, then:

- If FEAT_Debugv8p2 is implemented, this bit reads as 0.
- If FEAT_Debugv8p2 is not implemented, this bit might read as 0 or 1.

For more information, see 'EDPRSR.{DLK, SPD, PU} and the Core power domain'.

| SPD | Meaning |
|-----|--|
| 0b0 | If EDPRSR.PU is 0, it is not known whether the state of the debug registers in the Core power domain is lost. If EDPRSR.PU is 1, the state of the debug registers in the Core power domain has not been lost. |
| 0b1 | The state of the debug registers in the Core power domain has been lost. |

If the Core power domain is powered up, then, following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or DoubleLockStatus() == FALSE this bit clears to 0.
- If FEAT_DoubleLock is implemented and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether this bit clears to 0 or is unchanged.

When FEAT_DoPD is not implemented and the Core power domain is in either retention or powerdown state, the value of EDPRSR.SPD is IMPLEMENTATION DEFINED. For more information, see 'EDPRSR.SPD when the Core domain is in either retention or powerdown state'.

EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} and the Core power domain'.

This field is in the Core power domain.

The reset behavior of this field is:

- On a Cold reset, this field resets to 1.

Accessing this field has the following behavior:

- **RAZ/WI** if all of the following are true:
 - FEAT_DoPD is not implemented
 - !IsCorePowered()
- **UNKNOWN/WI** if all of the following are true:
 - IsCorePowered()
 - DoubleLockStatus()
- Otherwise, access to this field is **RO**.

PU, bit [0]

When FEAT_DoPD is implemented:

Core powerup status bit.

Access to this field is **RAO/WI**.

When FEAT_Debugv8p2 is implemented:

Core Powerup status bit. Indicates whether the debug registers in the Core power domain can be accessed.

| PU | Meaning |
|-----|---|
| 0b0 | Either the Core power domain is in a low-power or powerdown state, or FEAT_DoubleLock is implemented and DoubleLockStatus() == TRUE, meaning the debug registers in the Core power domain cannot be accessed. |
| 0b1 | The Core power domain is in a powerup state, and either FEAT_DoubleLock is not implemented or DoubleLockStatus() == FALSE, meaning the debug registers in the Core power domain can be accessed. |

If FEAT_DoubleLock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a CONSTRAINED UNPREDICTABLE value. For more information, see 'EDPRSR.{DLK, R} and reset state'

EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} and the Core power domain'

Access to this field is **RO**.

Otherwise:

Core Powerup status bit. Indicates whether the debug registers in the Core power domain can be accessed.

When the Core power domain is powered-up and DoubleLockStatus() == TRUE, then the value of EDPRSR.PU is IMPLEMENTATION DEFINED. See the description of the DLK bit for more information.

Otherwise, permitted values are:

| PU | Meaning |
|-----|---|
| 0b0 | Core power domain is in a low-power or powerdown state where the debug registers in the Core power domain cannot be accessed. |
| 0b1 | Core power domain is in a powerup state where the debug registers in the Core power domain can be accessed. |

If FEAT_DoubleLock is implemented, the Core power domain is powered up, and DoubleLockStatus() == TRUE, it is IMPLEMENTATION DEFINED whether this bit reads as 0 or 1.

If FEAT_DoubleLock is implemented, the PE is in reset state, and the PE entered reset state with the OS Double Lock locked this bit has a CONSTRAINED UNPREDICTABLE value. For more information see 'EDPRSR.{DLK, R} and reset state'

EDPRSR.{DLK, SPD, PU} describe whether registers in the Core power domain can be accessed, and whether their state has been lost since the last time the register was read. For more information, see 'EDPRSR.{DLK, SPD, PU} and the Core power domain'.

Access to this field is **RO**.

Accessing EDPRSR

On permitted accesses to the register, other access controls affect the behavior of some fields. See the field descriptions for more information.

If the Core power domain is powered up (EDPRSR.PU == 1), then following a read of EDPRSR:

- If FEAT_DoubleLock is not implemented or DoubleLockStatus() == FALSE, then:
 - EDPRSR.{SDR, SPMAD, SDAD, SPD} are cleared to 0.
 - EDPRSR.SR is cleared to 0 if the non-debug logic of the PE is not in reset state (EDPRSR.R == 0).
- If FEAT_DoubleLock is implemented and DoubleLockStatus() == TRUE, it is CONSTRAINED UNPREDICTABLE whether or not this clearing occurs.

If FEAT_DoPD is not implemented and the Core power domain is powered down (EDPRSR.PU == 0), then:

- EDPRSR.{SDR, SPMAD, SDAD, SR} are all UNKNOWN, and are either reset or restored on being powered up.
- EDPRSR.SPD is not cleared following a read of EDPRSR. See the SPD bit description for more information.

The clearing of bits is an indirect write to EDPRSR.

EDPRSR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0x314 | EDPRSR |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDRCR, External Debug Reserve Control Register

The EDRCR characteristics are:

Purpose

This register is used to allow imprecise entry to Debug state and clear sticky bits in [EDSCR](#).

Configuration

EDRCR is in the Core power domain.

Attributes

EDRCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|------|---|-----|---|------|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | CBRR0 | | CSPA | | CSE | | RES0 | | | | | | |

Bits [31:5]

Reserved, RES0.

CBRRQ, bit [4]

Allow imprecise entry to Debug state. The actions on writing to this bit are:

| CBRRQ | Meaning |
|-------|--|
| 0b0 | No action. |
| 0b1 | Allow imprecise entry to Debug state, for example by canceling pending bus accesses. |

Setting this bit to 1 allows a debugger to request imprecise entry to Debug state. An External Debug Request debug event must be pending before the debugger sets this bit to 1.

This feature is optional. If this feature is not implemented, writes to this bit are ignored.

CSPA, bit [3]

Clear Sticky Pipeline Advance. This bit is used to clear the [EDSCR](#).PipeAdv bit to 0. The actions on writing to this bit are:

| CSPA | Meaning |
|------|--|
| 0b0 | No action. |
| 0b1 | Clear the EDSCR .PipeAdv bit to 0. |

CSE, bit [2]

Clear Sticky Error. Used to clear the [EDSCR](#) cumulative error bits to 0. The actions on writing to this bit are:

| CSE | Meaning |
|-----|--|
| 0b0 | No action. |
| 0b1 | Clear the EDSCR .{TXU, RXO, ERR} bits, and, if the PE is in Debug state, the EDSCR .ITO bit, to 0. |

Bits [1:0]

Reserved, RES0.

Accessing EDRCR

EDRCR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0x090 | EDRCR |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **WI**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **WO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDSCR, External Debug Status and Control Register

The EDSCR characteristics are:

Purpose

Main control register for the debug implementation.

Configuration

External register EDSCR bits [30:29] are architecturally mapped to AArch64 System register [MDCCSR_EL0\[30:29\]](#).

EDSCR is in the Core power domain.

Attributes

EDSCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|--------|----|--------|----|-------|----|-----|----|---------|----|-----|----|--------|----|-----|----|-------|----|--------|----|-----|---|-----|---|-----|---|----|---|----|---|------|--|--------|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |
| TFO | | RXfull | | TXfull | | TORXO | | TXU | | PipeAdv | | ITE | | INTdis | | TDA | | MASC2 | | NSRES0 | | SDD | | NSE | | HDE | | RW | | EL | | AERR | | STATUS | |

TFO, bit [31]

When FEAT_TRF is implemented:

Trace Filter Override. Overrides the Trace Filter controls allowing the external debugger to trace any visible Exception level.

| TFO | Meaning |
|-----|--|
| 0b0 | Trace Filter controls are not affected. |
| 0b1 | Trace Filter controls in TRFCR_EL1 and TRFCR_EL2 are ignored. Trace Filter controls TRFCR and HTRFCR are ignored. |

When [OSLSR_EL1](#).OSLK is 1, this bit can be indirectly read and written through the [MDSCR_EL1](#) and [DBGDSCRext](#) System registers.

This bit is ignored by the PE when any of the following is true:

- ExternalSecureNoninvasiveDebugEnabled() is FALSE and the Effective value of [MDCR_EL3](#).STE is 1.
- FEAT_RME is implemented, ExternalRealmNoninvasiveDebugEnabled() is FALSE, and the Effective value of [MDCR_EL3](#).RLTE is 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

RXfull, bit [30]

DTRRX full.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Access to this field is **RO**.

TXfull, bit [29]

DTRTX full.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Access to this field is **RO**.

ITO, bit [28]

ITR overrun. Set to 0 on entry to Debug state.

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

RXO, bit [27]

DTRRX overrun.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Access to this field is **RO**.

TXU, bit [26]

DTRTX underrun.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Access to this field is **RO**.

PipeAdv, bit [25]

Pipeline Advance. Indicates that software execution is progressing.

| PipeAdv | Meaning |
|---------|---|
| 0b0 | No progress has been made by the PE since the last time this field was cleared to zero by writing 1 to EDRCR.CSPA . |
| 0b1 | Progress has been made by the PE since the last time this field was cleared to zero by writing 1 to EDRCR.CSPA . |

The architecture does not define precisely when this field is set to 1. It requires only that this happen periodically in Non-debug state to indicate that software execution is progressing. For example, a PE might set this field to 1 each time the PE retires one or more instructions, or at periodic intervals during the progression of an instruction.

When FEAT_MOPS is implemented, CPY, CPYF, SET, and SETG Memory Set and Copy instructions are examples of instructions that periodically make forward progress.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO**.

ITE, bit [24]

ITR empty.

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

INTdis, bits [23:22]

When FEAT_RME is implemented:

Interrupt disable. Disables taking interrupts in Non-debug state.

| INTdis | Meaning |
|--------|--|
| 0b00 | This bit has no effect on the masking of interrupts. |
| 0b01 | If ExternalInvasiveDebugEnabled() is TRUE, then all interrupts taken to Non-secure state are masked. If ExternalSecureInvasiveDebugEnabled() is TRUE, then all interrupts taken to Secure state are masked. If ExternalRootInvasiveDebugEnabled() is TRUE, then all interrupts taken to Root state are masked. If ExternalRealmInvasiveDebugEnabled() is TRUE, then all interrupts taken to Realm state are masked. |

Note

All interrupts includes virtual and SError interrupts.

When [OSLSR_EL1](#).OSLK is 1, this field can be indirectly read and written through the [MDSCR_EL1](#) and [DBGDSCRext](#) System registers.

The Effective value of this field is 0b00 when ExternalInvasiveDebugEnabled() is FALSE.

When FEAT_RME is implemented, bit[23] of this register is RES0.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

When FEAT_Debugv8p4 is implemented:

Interrupt disable. Disables taking interrupts in Non-debug state.

| INTdis | Meaning |
|--------|--|
| 0b00 | Masking of interrupts is controlled by PSTATE and interrupt routing controls. |
| 0b01 | If ExternalInvasiveDebugEnabled() is TRUE, then all interrupts taken to Non-secure state are masked. If ExternalSecureInvasiveDebugEnabled() is TRUE, then all interrupts taken to Secure state are masked. |

Note

All interrupts includes virtual and SError interrupts.

When [OSLSR_EL1](#).OSLK is 1, this field can be indirectly read and written through the [MDSCR_EL1](#) and [DBGDSCRext](#) System registers.

The Effective value of this field is 0b00 when ExternalInvasiveDebugEnabled() is FALSE.

When FEAT_Debugv8p4 is implemented, bit[23] of this register is RES0.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Interrupt disable. Disables taking interrupts in Non-debug state.

| INTdis | Meaning |
|--------|--|
| 0b00 | Masking of interrupts is controlled by PSTATE and interrupt routing controls. |
| 0b01 | If ExternalInvasiveDebugEnabled() is TRUE, then all interrupts taken to Non-secure EL1 are masked. |
| 0b10 | If ExternalInvasiveDebugEnabled() is TRUE, then all interrupts taken to Non-secure state are masked. If ExternalSecureInvasiveDebugEnabled() is TRUE, then all interrupts taken to Secure EL1 are masked. |
| 0b11 | If ExternalInvasiveDebugEnabled() is TRUE, then all interrupts taken to Non-secure state are masked. If ExternalSecureInvasiveDebugEnabled() is TRUE, then all interrupts taken to Secure state are masked. |

Note

All interrupts includes virtual and SError interrupts.

When [OSLSR_EL1](#).OSLK is 1, this field can be indirectly read and written through the [MDSCR_EL1](#) and [DBGDSCRExt](#) System registers.

The Effective value of this field is 0b00 when ExternalInvasiveDebugEnabled() is FALSE.

Support for the values 0b01 and 0b10 is IMPLEMENTATION DEFINED. If these values are not supported, they are reserved. If programmed with a reserved value, the PE behaves as if INTdis has been programmed with a defined value, other than for a direct read of EDSCR, and the value returned by a read of EDSCR.INTdis is UNKNOWN.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

TDA, bit [21]

Traps accesses to the following debug System registers:

- AArch64: [DBGBCR<n>_EL1](#), [DBGBVR<n>_EL1](#), [DBGWCR<n>_EL1](#), [DBGWVR<n>_EL1](#).
- AArch32: [DBGBCR<n>](#), [DBGBVR<n>](#), [DBGBXVR<n>](#), [DBGWCR<n>](#), [DBGWVR<n>](#).

| TDA | Meaning |
|-----|---|
| 0b0 | Accesses to debug System registers do not generate a Software Access Debug event. |
| 0b1 | Accesses to debug System registers generate a Software Access Debug event, if OSLSR_EL1 .OSLK is 0 and if halting is allowed. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

MA, bit [20]

Memory access mode. Controls the use of memory-access mode for accessing ITR and the DCC. This bit is ignored if in Non-debug state and set to zero on entry to Debug state.

Possible values of this field are:

| MA | Meaning |
|-----|---------------------|
| 0b0 | Normal access mode. |
| 0b1 | Memory access mode. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

SC2, bit [19]

When FEAT_PCSRv8 is implemented, (FEAT_VHE is implemented or FEAT_Debugv8p2 is implemented) and FEAT_PCSRv8p2 is not implemented:

Sample [CONTEXTIDR_EL2](#). Controls whether the PC Sample-based Profiling Extension samples [CONTEXTIDR_EL2](#) or [VTTBR_EL2](#).VMID.

| SC2 | Meaning |
|-----|---|
| 0b0 | Sample VTTBR_EL2 .VMID. |
| 0b1 | Sample CONTEXTIDR_EL2 . |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Otherwise:

Reserved, RES0.

NS, bit [18]

When FEAT_RME is implemented:

Non-secure status. Together with the NSE field, gives the current Security state:

| NSE | NS | Meaning |
|-----|-----|-------------|
| 0b0 | 0b0 | Secure. |
| 0b0 | 0b1 | Non-secure. |
| 0b1 | 0b0 | Root. |
| 0b1 | 0b1 | Realm. |

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

Otherwise:

Non-secure status. In Debug state, gives the current Security state:

| NS | Meaning |
|-----|-------------------|
| 0b0 | Secure state. |
| 0b1 | Non-secure state. |

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

Bit [17]

Reserved, RES0.

SDD, bit [16]

When FEAT_RME is implemented:

Secure debug disabled.

Reports the inverse of ExternalRootInvasiveDebugEnabled().

Access to this field is **RO**.

Otherwise:

Secure debug disabled.

On entry to Debug state:

- If entering in Secure state, SDD is set to 0.
- If entering in Non-secure state, SDD is set to the inverse of ExternalSecureInvasiveDebugEnabled().

In Debug state, the value of the SDD bit does not change, even if ExternalSecureInvasiveDebugEnabled() changes.

In Non-debug state:

- SDD returns the inverse of ExternalSecureInvasiveDebugEnabled(). If the authentication signals that control ExternalSecureInvasiveDebugEnabled() change, a context synchronization event is required to guarantee their effect.
- This bit is unaffected by the Security state of the PE.

If EL3 is not implemented and the implementation is Non-secure, this bit is RES1.

Access to this field is **RO**.

NSE, bit [15]

When FEAT_RME is implemented:

Together with the NS field, this field gives the current Security state.

For a description of the values derived by evaluating NS and NSE together, see EDSCR.NS.

In Non-debug state, this bit is UNKNOWN.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

HDE, bit [14]

Halting debug enable.

| HDE | Meaning |
|------------|--|
| 0b0 | Halting disabled for Breakpoint, Watchpoint and Halt Instruction debug events. |
| 0b1 | Halting enabled for Breakpoint, Watchpoint and Halt Instruction debug events. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

RW, bits [13:10]

Exception level Execution state status. In Debug state, each bit gives the current Execution state of each Exception level.

| RW | Meaning | Applies when |
|--------|--|--|
| 0b1111 | Any of the following: <ul style="list-style-type: none"> The PE is in Non-debug state. The PE is at EL0 using AArch64. The PE is not at EL0, and EL1, EL2, and EL3 are using AArch64. | |
| 0b1110 | The PE is in Debug state at EL0. EL0 is using AArch32. EL1, EL2, and EL3 are using AArch64. | When AArch32 is supported |
| 0b110x | The PE is in Debug state. EL0 and EL1 are using AArch32. EL2 is enabled in the current Security state and is using AArch64. If implemented, EL3 is using AArch64. | When AArch32 is supported and EL2 is implemented |
| 0b10xx | The PE is in Debug state. EL0 and EL1 are using AArch32. EL2 is not implemented, disabled in the current Security state, or using AArch32. EL3 is using AArch64. | When AArch32 is supported and EL3 is implemented |
| 0b0xxx | The PE is in Debug state. All Exception levels are using AArch32. | When AArch32 is supported |

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **RAO/WI**.
- Otherwise, access to this field is **RO**.

EL, bits [9:8]

Exception level. In Debug state, gives the current Exception level of the PE.

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RO**.

A, bit [7]

SError interrupt pending. In Debug state, indicates whether an SError interrupt is pending:

- If [HCR_EL2](#).{AMO, TGE} = {1, 0}, EL2 is enabled in the current Security state, and the PE is executing at EL0 or EL1, a virtual SError interrupt.
- Otherwise, a physical SError interrupt.

| A | Meaning |
|-----|------------------------------|
| 0b0 | No SError interrupt pending. |
| 0b1 | SErrror interrupt pending. |

A debugger can read EDSCR to check whether an SError interrupt is pending without having to execute further instructions. A pending SError might indicate data from target memory is corrupted.

Accessing this field has the following behavior:

- When the PE is in Non-debug state, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

ERR, bit [6]

Cumulative error flag. This bit is set to 1 following exceptions in Debug state and on any signaled overrun or underrun on the DTR or EDITR.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Access to this field is **RO**.

STATUS, bits [5:0]

Debug status flags.

| STATUS | Meaning |
|---------------|--|
| 0b000001 | PE is restarting, exiting Debug state. |
| 0b000010 | PE is in Non-debug state. |
| 0b000111 | Breakpoint. |
| 0b010011 | External debug request. |
| 0b011011 | Halting step, normal. |
| 0b011111 | Halting step, exclusive. |
| 0b100011 | OS Unlock Catch. |
| 0b100111 | Reset Catch. |
| 0b101011 | Watchpoint. |
| 0b101111 | HLT instruction. |
| 0b110011 | Software access to debug register. |
| 0b110111 | Exception Catch. |
| 0b111011 | Halting step, no syndrome. |

All other values of STATUS are reserved.

Access to this field is **RO**.

Accessing EDSCR

EDSCR can be accessed through the external debug interface:

| Component | Offset | Instance |
|------------------|---------------|-----------------|
| Debug | 0x088 | EDSCR |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDVIDSR, External Debug Virtual Context Sample Register

The EDVIDSR characteristics are:

Purpose

Contains sampled values captured on reading [EDPCSR](#)[31:0].

Configuration

EDVIDSR is in the Core power domain.

This register is present only when FEAT_PCSRv8 is implemented and FEAT_PCSRv8p2 is not implemented. Otherwise, direct accesses to EDVIDSR are RES0.

If FEAT_VHE is implemented, the format of this register differs depending on the value of [EDSCR](#).SC2.

Implemented only if the OPTIONAL PC Sample-based Profiling Extension is implemented in the external debug registers space.

When the PC Sample-based Profiling Extension is implemented in the external debug registers space, if EL2 is not implemented and EL3 is not implemented, it is IMPLEMENTATION DEFINED whether EDVIDSR is implemented.

Note

FEAT_PCSRv8p2 implements the PC Sample-based Profiling Extension in the Performance Monitors registers space.

Attributes

EDVIDSR is a 32-bit register.

Field descriptions

When FEAT_VHE is not implemented or EDSCR.SC2 == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|---|---|------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NS | E2 | E3 | HV | RES0 | | | | | | | | | | | | VMID[15:8] | | | | | | | | VMID | | | | | | | |

This format applies in all Armv8.0 implementations.

NS, bit [31]

Non-secure state sample. Indicates the Security state associated with the most recent [EDPCSR](#) sample.

If EL3 is not implemented, this bit indicates the Effective value of SCR.NS.

| NS | Meaning |
|-----|----------------------------------|
| 0b0 | Sample is from Secure state. |
| 0b1 | Sample is from Non-secure state. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

E2, bit [30]**When EL2 is implemented:**

Exception level 2 status sample. Indicates whether the most recent [EDPCSR](#) sample was associated with EL2.

| E2 | Meaning |
|-----------|-------------------------|
| 0b0 | Sample is not from EL2. |
| 0b1 | Sample is from EL2. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

E3, bit [29]**When EL3 is implemented and AArch64 is supported:**

Exception level 3 status sample. Indicates whether the most recent [EDPCSR](#) sample was associated with EL3 using AArch64.

| E3 | Meaning |
|-----------|---------------------------------------|
| 0b0 | Sample is not from EL3 using AArch64. |
| 0b1 | Sample is from EL3 using AArch64. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

HV, bit [28]

EDPCSRhi ([EDPCSR](#)[63:32]) valid. Indicates whether bits [63:32] of the most recent [EDPCSR](#) sample might be nonzero:

| HV | Meaning |
|-----------|--|
| 0b0 | Bits[63:32] of the most recent EDPCSR sample are zero. |
| 0b1 | Bits[63:32] of the most recent EDPCSR sample might be nonzero. |

An EDVIDSR.HV value of 1 does not mean that the value of EDPCSRhi is nonzero. An EDVIDSR.HV value of 0 is a hint that EDPCSRhi ([EDPCSR](#)[63:32]) does not need to be read.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Bits [27:16]

Reserved, RES0.

VMID[15:8], bits [15:8]**When FEAT_VMID16 is implemented and EL2 is implemented:**

Extension to VMID[7:0]. For more information, see VMID[7:0].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID, bits [7:0]**When EL2 is implemented:**

VMID sample. The VMID associated with the most recent EDPCSRlo ([EDPCSR](#)[31:0]) sample. When the most recent [EDPCSR](#) sample was generated:

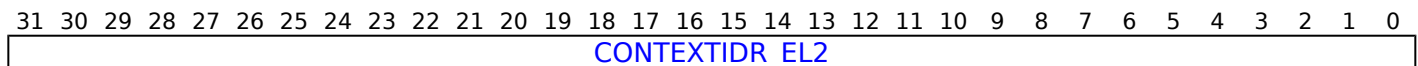
- This field is RES0 if any of the following apply:
 - The PE is executing in Secure state.
 - The PE is executing at EL2.
- Otherwise:
 - If EL2 is using AArch64 and either FEAT_VMID16 is not implemented or [VTCR_EL2](#).VS is 1, this field is set to [VTTBR_EL2](#).VMID.
 - If EL2 is using AArch64, FEAT_VMID16 is implemented, and [VTCR_EL2](#).VS is 0, PMVIDSR.VMID[7:0] is set to [VTTBR_EL2](#).VMID[7:0] and PMVIDSR.VMID[15:8] is RES0.
 - If EL2 is using AArch32, this field is set to [VTTBR](#).VMID.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

When (FEAT_VHE is implemented or FEAT_Debugv8p2 is implemented) and EDSCR.SC2 == 1:**CONTEXTIDR_EL2, bits [31:0]**

Context ID. The value of [CONTEXTIDR_EL2](#) that is associated with the most recent [EDPCSR](#) sample. When the most recent [EDPCSR](#) sample is generated:

- If the PE is not executing at EL3, EL2 is using AArch64, and EL2 is enabled in the current Security state, then this field is set to the Context ID sampled from [CONTEXTIDR_EL2](#).
- Otherwise, this field is set to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing EDVIDSR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

EDVIDSR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0x0A8 | EDVIDSR |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

EDWAR, External Debug Watchpoint Address Register

The EDWAR characteristics are:

Purpose

Returns the virtual data address being accessed when a Watchpoint Debug Event was triggered.

Configuration

EDWAR is in the Core power domain.

Attributes

EDWAR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Watchpoint address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Watchpoint address | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Watchpoint address. The data virtual address being accessed when a Watchpoint Debug Event was triggered and caused entry to Debug state. This address must be within a naturally-aligned block of memory of power-of-two size no larger than the [DC ZVA](#) block size.

The value of this register is UNKNOWN if the PE is in Non-debug state, or if Debug state was entered other than for a Watchpoint debug event.

The value of EDWAR[63:32] is UNKNOWN if Debug state was entered for a Watchpoint debug event taken from AArch32 state.

The EDWAR is subject to the same alignment rules as the reporting of a watchpointed address in the FAR. See 'Determining the memory location that caused a Watchpoint exception'.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing EDWAR

EDWAR can be accessed through the external debug interface:

| Component | Offset | Instance | Range |
|-----------|--------|----------|-------|
| Debug | 0x030 | EDWAR | 31:0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

| Component | Offset | Instance | Range |
|-----------|--------|----------|-------|
| Debug | 0x034 | EDWAR | 63:32 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRCIDR0, Component Identification Register 0

The ERRCIDR0 characteristics are:

Purpose

Provides discovery information about the component.
For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.
ERRCIDR0 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCIDR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | PRMBL_0 | | | | | | | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Component identification preamble, segment 0.
Reads as 0x0D.
Access to this field is **RO**.

Accessing ERRCIDR0

ERRCIDR0 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| RAS | 0xFF0 | ERRCIDR0 |

Accesses on this interface are **RO**.

ERRCIDR1, Component Identification Register 1

The ERRCIDR1 characteristics are:

Purpose

Provides discovery information about the component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

ERRCIDR1 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCIDR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---|---------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | CLASS | | | | PRMBL_1 | | | |

Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

| CLASS | Meaning |
|--------|---|
| 0b1111 | Generic peripheral with IMPLEMENTATION DEFINED register layout. |

Other values are defined by the CoreSight Architecture.

This field reads as 0xF.

PRMBL_1, bits [3:0]

Component identification preamble, segment 1.

Reads as 0b0000.

Access to this field is **RO**.

Accessing ERRCIDR1

ERRCIDR1 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| RAS | 0xFF4 | ERRCIDR1 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRCIDR2, Component Identification Register 2

The ERRCIDR2 characteristics are:

Purpose

Provides discovery information about the component.
For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.
ERRCIDR2 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCIDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | PRMBL_2 | | | | | | | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Component identification preamble, segment 2.
Reads as 0x05.
Access to this field is **RO**.

Accessing ERRCIDR2

ERRCIDR2 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| RAS | 0xFF8 | ERRCIDR2 |

Accesses on this interface are **RO**.

ERRCIDR3, Component Identification Register 3

The ERRCIDR3 characteristics are:

Purpose

Provides discovery information about the component.
For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.
ERRCIDR3 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCIDR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | PRMBL_3 | | | | | | | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Component identification preamble, segment 3.
Reads as 0xB1.
Access to this field is **RO**.

Accessing ERRCIDR3

ERRCIDR3 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| RAS | 0xFFC | ERRCIDR3 |

Accesses on this interface are **RO**.

ERRCRICR0, Critical Error Interrupt Configuration Register 0

The ERRCRICR0 characteristics are:

Purpose

Critical Error Interrupt configuration register.

Configuration

This register is present only when (the Critical Error Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR<n> registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRCRICR0 are RES0.

ERRCRICR0 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCRICR0 is a 64-bit register.

Field descriptions

When the Critical Error Interrupt is implemented and the implementation uses the recommended layout for the ERRIRQCR<n> registers:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | ADDR | | | | | | | | | | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | RES0 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:56]

Reserved, RES0.

ADDR, bits [55:2]

Message Signaled Interrupt address. (ERRCRICR0.ADDR << 2) is the address that the component writes to when signaling the Critical Error Interrupt. Bits [1:0] of the address are always zero.

The physical address size supported by the component is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When the implementation does not use the recommended layout for the ERRIRQCR<n> registers:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRCRICR0

ERRCRICR0 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-----------|
| RAS | 0xEA0 | ERRCRICR0 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRCRICR1, Critical Error Interrupt Configuration Register 1

The ERRCRICR1 characteristics are:

Purpose

Critical Error Interrupt configuration register.

Configuration

This register is present only when (the Critical Error Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR<n> registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRCRICR1 are RES0.

ERRCRICR1 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCRICR1 is a 32-bit register.

Field descriptions

When the Critical Error Interrupt is implemented and the implementation uses the recommended layout for the ERRIRQCR<n> registers:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| DATA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

DATA, bits [31:0]

Payload for the message signaled interrupt.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

When the implementation does not use the recommended layout for the ERRIRQCR<n> registers:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRCRICR1

ERRCRICR1 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-----------|
| RAS | 0xEA8 | ERRCRICR1 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRCRICR2, Critical Error Interrupt Configuration Register 2

The ERRCRICR2 characteristics are:

Purpose

Critical Error Interrupt control and configuration register.

Configuration

This register is present only when (the Critical Error Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR<n> registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRCRICR2 are RES0.

ERRCRICR2 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRCRICR2 is a 32-bit register.

Field descriptions

When the Critical Error Interrupt is implemented and the implementation uses the recommended layout for the ERRIRQCR<n> registers:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|-------|---|----|---|---------|---|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | IRQEN | | NSMSI | | SH | | MemAttr | | |

Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

When the component supports disabling message signaled interrupts:

Message signaled interrupt enable. Enables generation of message signaled interrupts.

| IRQEN | Meaning |
|-------|-----------|
| 0b0 | Disabled. |
| 0b1 | Enabled. |

The reset behavior of this field is:

- On an Error recovery reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Message signaled interrupt enable.

Message signaled interrupts are always enabled.

NSMSI, bit [6]

When the component supports configuring the Security attribute for message signaled interrupts and the component does not allow Non-secure writes to ERRCRICR2:

Security attribute. Defines the physical address space for message signaled interrupts.

| NSMSI | Meaning |
|-------|-------------|
| 0b0 | Secure. |
| 0b1 | Non-secure. |

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

When the component allows Non-secure writes to ERRCRICR2:

Reserved, RES0.

Security attribute. Defines the physical address space for message signaled interrupts.

The Security attribute used for message signaled interrupts is Non-secure.

Otherwise:

Reserved, RES0.

Security attribute. Defines the physical address space for message signaled interrupts.

The Security attribute for message signaled interrupts is IMPLEMENTATION DEFINED.

SH, bits [5:4]

When the component supports configuring the Shareability domain for message signaled interrupts:

Shareability. Defines the Shareability domain for message signaled interrupts.

| SH | Meaning |
|------|------------------|
| 0b00 | Not shared. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

All other values are reserved.

This field is ignored when ERRCRICR2.MemAttr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Shareability.

The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

MemAttr, bits [3:0]

When the component supports configuring the memory type for message signaled interrupts:

Memory type. Defines the memory type and attributes for message signaled interrupts.

| MemAttr | Meaning |
|---------|--|
| 0b0000 | Device-nGnRnE memory. |
| 0b0001 | Device-nGnRE memory. |
| 0b0010 | Device-nGRE memory. |
| 0b0011 | Device-GRE memory. |
| 0b0101 | Normal memory, Inner Non-cacheable, Outer Non-cacheable. |
| 0b0110 | Normal memory, Inner Write-Through, Outer Non-cacheable. |
| 0b0111 | Normal memory, Inner Write-Back, Outer Non-cacheable. |
| 0b1001 | Normal memory, Inner Non-cacheable, Outer Write-Through. |
| 0b1010 | Normal memory, Inner Write-Through, Outer Write-Through. |
| 0b1011 | Normal memory, Inner Write-Back, Outer Write-Through. |
| 0b1101 | Normal memory, Inner Non-cacheable, Outer Write-Back. |
| 0b1110 | Normal memory, Inner Write-Through, Outer Write-Back. |
| 0b1111 | Normal memory, Inner Write-Back, Outer Write-Back. |

All other values are reserved.

Note

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Memory type.

The memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

When the implementation does not use the recommended layout for the ERRIRQCR<n> registers:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRCRICR2

ERRCRICR2 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-----------|
| RAS | 0xEAC | ERRCRICR2 |

Accesses on this interface are **RW**.

ERRDEVAFF, Device Affinity Register

The ERRDEVAFF characteristics are:

Purpose

For a group of error records that has affinity with a single PE or a group of PEs, ERRDEVAFF is a copy of [MPIDR_EL1](#) or part of [MPIDR_EL1](#):

- If the group of error records has affinity with a single PE, the affinity level is 0, ERRDEVAFF reads the same value as [MPIDR_EL1](#), and ERRDEVAFF.F0V reads-as-one to indicate affinity level 0.
- If the group of error records has affinity with a group of PEs, the affinity level is 1, 2, or 3, parts of ERRDEVAFF reads the same value as parts of [MPIDR_EL1](#), and the rest of ERRDEVAFF indicates the level.

For example, if the group of PEs is a subset of the PEs at affinity level 1 then all of the following are true:

- All the PEs in the group have the same values in [MPIDR_EL1](#).{Aff3,Aff2}, and these values are equal to ERRDEVAFF.{Aff3,Aff2}.
- ERRDEVAFF.Aff1 is nonzero and not 0x80, and ERRDEVAFF.{Aff0,F0V} read-as-zero, to indicate at least affinity level 1. The subset of PEs at level 1 that the group of error records has affinity with is indicated by the least-significant set bit in ERRDEVAFF.Aff1. In this example, if ERRDEVAFF.Aff1[2:0] is 0b100, then the group of error records has affinity with the up-to 8 PEs that have [MPIDR_EL1](#).Aff1[7:3] == ERRDEVAFF.Aff1[7:3].

If RAS System Architecture v1.1 is not implemented, ERRDEVAFF can only describe a group of error records that is affine with a single PE or all the PEs at an affinity level.

Configuration

This register is present only when the group of error records has affinity with a PE or cluster of PEs. Otherwise, direct accesses to ERRDEVAFF are RES0.

ERRDEVAFF is implemented only as part of a memory-mapped group of error records.

Attributes

ERRDEVAFF is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|------|----|----|----|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|------|------|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | Aff3 | | | | | | | | |
| FOV | U | RES0 | | | | | | MT | Aff2 | | | | | | | | Aff1 | | | | | | | | Aff0 | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

PE affinity level 3. The [MPIDR_EL1](#).Aff3 field, viewed from the highest Exception level of the associated PE or PEs.

F0V, bit [31]

Indicates that the ERRDEVAFF.Aff0 field is valid.

| F0V | Meaning |
|------------|---|
| 0b0 | ERRDEVAFF.Aff0 is not valid, and the PE affinity is above level 0 or a subset of level 0. |
| 0b1 | ERRDEVAFF.Aff0 is valid, and the PE affinity is at level 0. |

U, bit [30]

When **ERRDEVAFF.F0V == 1**:

Uniprocessor. The [MPIDR_EL1](#).U field, viewed from the highest Exception level of the associated PE.

Otherwise:

Reserved, UNKNOWN.

Bits [29:25]

Reserved, RES0.

MT, bit [24]

When **ERRDEVAFF.F0V == 1**:

Multithreaded. The [MPIDR_EL1](#).MT field, viewed from the highest Exception level of the associated PE.

Otherwise:

Reserved, UNKNOWN.

Aff2, bits [23:16]

When affine with a PE or PEs at affinity level 2 or below:

PE affinity level 2. The [MPIDR_EL1](#).Aff2 field, viewed from the highest Exception level of the associated PE or PEs.

When affine with a sub-set of PEs at affinity level 2:

PE affinity level 2. Defines part of the [MPIDR_EL1](#).Aff2 field, viewed from the highest Exception level of the associated PEs.

| Aff2 | Meaning |
|-------------|--|
| 0bxxxxxxx1 | ERRDEVAFF.Aff2[7:1] is the value of MPIDR_EL1 .Aff2[7:1], viewed from the highest Exception level of the associated PEs. |
| 0bxxxxxx10 | ERRDEVAFF.Aff2[7:2] is the value of MPIDR_EL1 .Aff2[7:2], viewed from the highest Exception level of the associated PEs. |
| 0bxxxxx100 | ERRDEVAFF.Aff2[7:3] is the value of MPIDR_EL1 .Aff2[7:3], viewed from the highest Exception level of the associated PEs. |
| 0bxxxx1000 | ERRDEVAFF.Aff2[7:4] is the value of MPIDR_EL1 .Aff2[7:4], viewed from the highest Exception level of the associated PEs. |
| 0bxxx10000 | ERRDEVAFF.Aff2[7:5] is the value of MPIDR_EL1 .Aff2[7:5], viewed from the highest Exception level of the associated PEs. |
| 0bxx100000 | ERRDEVAFF.Aff2[7:6] is the value of MPIDR_EL1 .Aff2[7:6], viewed from the highest Exception level of the associated PEs. |
| 0bx1000000 | ERRDEVAFF.Aff2[7] is the value of MPIDR_EL1 .Aff2[7], viewed from the highest Exception level of the associated PEs. |

Otherwise:

PE affinity level 2. Indicates whether the PE affinity is at level 3.

| Aff2 | Meaning |
|------|----------------------------|
| 0x80 | PE affinity is at level 3. |

All other values are reserved.

Aff1, bits [15:8]**When affine with a PE or PEs at affinity level 1 or below:**

PE affinity level 1. The [MPIDR_EL1](#).Aff1 field, viewed from the highest Exception level of the associated PE or PEs.

When affine with a sub-set of PEs at affinity level 1:

PE affinity level 1. Defines part of the [MPIDR_EL1](#).Aff1 field, viewed from the highest Exception level of the associated PEs.

| Aff1 | Meaning |
|------------|--|
| 0bxxxxxxx1 | ERRDEVAFF.Aff1[7:1] is the value of MPIDR_EL1 .Aff1[7:1], viewed from the highest Exception level of the associated PEs. |
| 0bxxxxxx10 | ERRDEVAFF.Aff1[7:2] is the value of MPIDR_EL1 .Aff1[7:2], viewed from the highest Exception level of the associated PEs. |
| 0bxxxxx100 | ERRDEVAFF.Aff1[7:3] is the value of MPIDR_EL1 .Aff1[7:3], viewed from the highest Exception level of the associated PEs. |
| 0bxxxx1000 | ERRDEVAFF.Aff1[7:4] is the value of MPIDR_EL1 .Aff1[7:4], viewed from the highest Exception level of the associated PEs. |
| 0bxxx10000 | ERRDEVAFF.Aff1[7:5] is the value of MPIDR_EL1 .Aff1[7:5], viewed from the highest Exception level of the associated PEs. |
| 0bxx100000 | ERRDEVAFF.Aff1[7:6] is the value of MPIDR_EL1 .Aff1[7:6], viewed from the highest Exception level of the associated PEs. |
| 0bx1000000 | ERRDEVAFF.Aff1[7] is the value of MPIDR_EL1 .Aff1[7], viewed from the highest Exception level of the associated PEs. |

Otherwise:

PE affinity level 1. Indicates whether the PE affinity is at level 2.

| Aff1 | Meaning |
|------|--|
| 0x00 | PE affinity is above level 2 or a subset of level 2. |
| 0x80 | PE affinity is at level 2. |

Aff0, bits [7:0]**When affine with a PE at affinity level 0:**

PE affinity level 0. The [MPIDR_EL1](#).Aff0 field, viewed from the highest Exception level of the associated PE.

When affine with a sub-set of PEs at affinity level 0:

PE affinity level 0. Defines part of the [MPIDR_EL1](#).Aff0 field, viewed from the highest Exception level of the associated PEs.

| Aff0 | Meaning |
|------------|--|
| 0bxxxxxxx1 | ERRDEVAFF.Aff0[7:1] is the value of MPIDR_EL1 .Aff0[7:1], viewed from the highest Exception level of the associated PEs. |
| 0bxxxxxx10 | ERRDEVAFF.Aff0[7:2] is the value of MPIDR_EL1 .Aff0[7:2], viewed from the highest Exception level of the associated PEs. |
| 0bxxxxx100 | ERRDEVAFF.Aff0[7:3] is the value of MPIDR_EL1 .Aff0[7:3], viewed from the highest Exception level of the associated PEs. |
| 0bxxxx1000 | ERRDEVAFF.Aff0[7:4] is the value of MPIDR_EL1 .Aff0[7:4], viewed from the highest Exception level of the associated PEs. |
| 0bxxx10000 | ERRDEVAFF.Aff0[7:5] is the value of MPIDR_EL1 .Aff0[7:5], viewed from the highest Exception level of the associated PEs. |
| 0bxx100000 | ERRDEVAFF.Aff0[7:6] is the value of MPIDR_EL1 .Aff0[7:6], viewed from the highest Exception level of the associated PEs. |
| 0bx1000000 | ERRDEVAFF.Aff0[7] is the value of MPIDR_EL1 .Aff0[7], viewed from the highest Exception level of the associated PEs. |

Otherwise:

PE affinity level 0. Indicates whether the PE affinity is at level 1.

| Aff0 | Meaning |
|------|--|
| 0x00 | PE affinity is above level 1 or a subset of level 1. |
| 0x80 | PE affinity is at level 1. |

Accessing ERRDEVAFF

ERRDEVAFF can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-----------|
| RAS | 0xFA8 | ERRDEVAFF |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRDEVARCH, Device Architecture Register

The ERRDEVARCH characteristics are:

Purpose

Provides discovery information for the component.

Configuration

ERRDEVARCH is implemented only as part of a memory-mapped group of error records.

Attributes

ERRDEVARCH is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|---------|----------|----|----|----|---------|----|----|----|----------|----|---|---|---|---|---|---|---|---|---|---|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| ARCHITECT | | | | | | | | | | | PRESENT | REVISION | | | | ARCHVER | | | | ARCHPART | | | | | | | | | | | | |

ARCHITECT, bits [31:21]

Architect. Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code.

| ARCHITECT | Meaning |
|---------------|--|
| 0b01000111011 | JEP106 continuation code 0x4, ID code 0x3B. Arm Limited. |

Other values are defined by the JEDEC JEP106 standard.

This field reads as 0x23B.

PRESENT, bit [20]

DEVARCH Present. Defines that the DEVARCH register is present.

| PRESENT | Meaning |
|---------|--|
| 0b0 | Device Architecture information not present. |
| 0b1 | Device Architecture information present. |

This field reads as 1.

REVISION, bits [19:16]

Revision. Defines the architecture revision of the component.

| REVISION | Meaning |
|----------|---|
| 0b0000 | RAS System Architecture v1.0. |
| 0b0001 | RAS System Architecture v1.1. As 0b0000 and also: <ul style="list-style-type: none"> Simplifies ERR<n>STATUS. Adds support for additional ERR<n>MISC<m> registers. Adds support for the optional RAS Timestamp Extension. Adds support for the optional Common Fault Injection Model Extension. |

All other values are reserved.

ARCHVER, bits [15:12]

Architecture Version. Defines the architecture version of the component.

| ARCHVER | Meaning |
|---------|-----------------------------|
| 0b0000 | RAS System Architecture v1. |

All other values are reserved.

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHVER is ARCHID[15:12].

This field reads as 0b0000.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

| ARCHPART | Meaning |
|----------|--------------------------|
| 0xA00 | RAS System Architecture. |

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHPART is ARCHID[11:0].

This field reads as 0xA00.

Accessing ERRDEVARCH

ERRDEVARCH can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|------------|
| RAS | 0xFBC | ERRDEVARCH |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRDEVID, Device Configuration Register

The ERRDEVID characteristics are:

Purpose

Provides discovery information for the component.

Configuration

ERRDEVID is implemented only as part of a memory-mapped group of error records.

Attributes

ERRDEVID is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | NUM | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

NUM, bits [15:0]

Highest numbered index of the error records in this group, plus one. Each implemented record is owned by a node. A node might own multiple records.

This manual describes a group of error records accessed via a standard 4KB memory-mapped peripheral. For a 4KB peripheral, up to 24 error records can be accessed if the Common Fault Injection Model is implemented, and up to 56 otherwise.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRDEVID

ERRDEVID can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| RAS | 0xFC8 | ERRDEVID |

Accesses on this interface are **RO**.

ERRERICR0, Error Recovery Interrupt Configuration Register 0

The ERRERICR0 characteristics are:

Purpose

Error Recovery Interrupt configuration register.

Configuration

This register is present only when (the Error Recovery Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR<n> registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRERICR0 are RES0.

ERRERICR0 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRERICR0 is a 64-bit register.

Field descriptions

When the Error Recovery Interrupt is implemented and the implementation uses the recommended layout for the ERRIRQCR<n> registers:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | ADDR | | | | | | | | | | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | RES0 | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:56]

Reserved, RES0.

ADDR, bits [55:2]

Message Signaled Interrupt address. (ERRERICR0.ADDR << 2) is the address that the component writes to when signaling the Error Recovery Interrupt. Bits [1:0] of the address are always zero.

The physical address size supported by the component is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When the implementation does not use the recommended layout for the ERRIRQCR<n> registers:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRERICR0

ERRERICR0 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-----------|
| RAS | 0xE90 | ERRERICR0 |

Accesses on this interface are **RW**.

ERRERICR1, Error Recovery Interrupt Configuration Register 1

The ERRERICR1 characteristics are:

Purpose

Error Recovery Interrupt configuration register.

Configuration

This register is present only when (the Error Recovery Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR<n> registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRERICR1 are RES0.

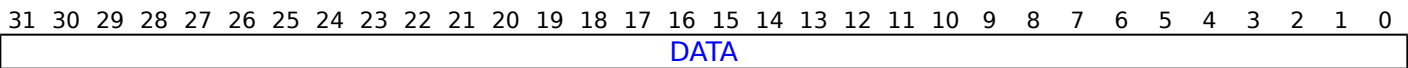
ERRERICR1 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRERICR1 is a 32-bit register.

Field descriptions

When the Error Recovery Interrupt is implemented and the implementation uses the recommended layout for the ERRIRQCR<n> registers:



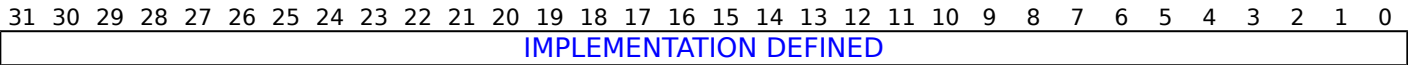
DATA, bits [31:0]

Payload for the message signaled interrupt.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

When the implementation does not use the recommended layout for the ERRIRQCR<n> registers:



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRERICR1

ERRERICR1 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-----------|
| RAS | 0xE98 | ERRERICR1 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRERICR2, Error Recovery Interrupt Configuration Register 2

The ERRERICR2 characteristics are:

Purpose

Error Recovery Interrupt control and configuration register.

Configuration

This register is present only when (the Error Recovery Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR<n> registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRERICR2 are RES0.

ERRERICR2 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRERICR2 is a 32-bit register.

Field descriptions

When the Error Recovery Interrupt is implemented and the implementation uses the recommended layout for the ERRIRQCR<n> registers:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|---|-------|---|-------|---|----|---------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | RES0 | | | | | | | | | | | | IRQEN | | NSMSI | | SH | MemAttr | | | |

Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

When the component supports disabling message signaled interrupts:

Message signaled interrupt enable. Enables generation of message signaled interrupts.

| IRQEN | Meaning |
|-------|-----------|
| 0b0 | Disabled. |
| 0b1 | Enabled. |

The reset behavior of this field is:

- On an Error recovery reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Message signaled interrupt enable.

Message signaled interrupts are always enabled.

NSMSI, bit [6]

When the component supports configuring the Security attribute for message signaled interrupts and the component does not allow Non-secure writes to ERRERICR2:

Security attribute. Defines the physical address space for message signaled interrupts.

| NSMSI | Meaning |
|-------|-------------|
| 0b0 | Secure. |
| 0b1 | Non-secure. |

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

When the component allows Non-secure writes to ERRERICR2:

Reserved, RES0.

Security attribute. Defines the physical address space for message signaled interrupts.

The Security attribute used for message signaled interrupts is Non-secure.

Otherwise:

Reserved, RES0.

Security attribute. Defines the physical address space for message signaled interrupts.

The Security attribute for message signaled interrupts is IMPLEMENTATION DEFINED.

SH, bits [5:4]

When the component supports configuring the Shareability domain for message signaled interrupts:

Shareability. Defines the Shareability domain for message signaled interrupts.

| SH | Meaning |
|------|------------------|
| 0b00 | Not shared. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

All other values are reserved.

This field is ignored when ERRERICR2.MemAttr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Shareability.

The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

MemAttr, bits [3:0]

When the component supports configuring the memory type for message signaled interrupts:

Memory type. Defines the memory type and attributes for message signaled interrupts.

| MemAttr | Meaning |
|---------|--|
| 0b0000 | Device-nGnRnE memory. |
| 0b0001 | Device-nGnRE memory. |
| 0b0010 | Device-nGRE memory. |
| 0b0011 | Device-GRE memory. |
| 0b0101 | Normal memory, Inner Non-cacheable, Outer Non-cacheable. |
| 0b0110 | Normal memory, Inner Write-Through, Outer Non-cacheable. |
| 0b0111 | Normal memory, Inner Write-Back, Outer Non-cacheable. |
| 0b1001 | Normal memory, Inner Non-cacheable, Outer Write-Through. |
| 0b1010 | Normal memory, Inner Write-Through, Outer Write-Through. |
| 0b1011 | Normal memory, Inner Write-Back, Outer Write-Through. |
| 0b1101 | Normal memory, Inner Non-cacheable, Outer Write-Back. |
| 0b1110 | Normal memory, Inner Write-Through, Outer Write-Back. |
| 0b1111 | Normal memory, Inner Write-Back, Outer Write-Back. |

All other values are reserved.

Note

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

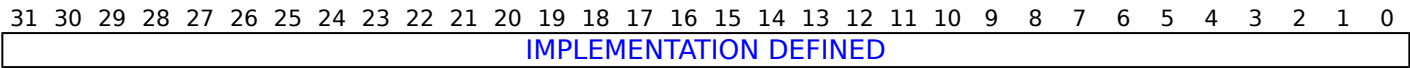
Otherwise:

Reserved, RES0.

Memory type.

The memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

When the implementation does not use the recommended layout for the ERRIRQCR<n> registers:



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRERICR2

ERRERICR2 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-----------|
| RAS | 0xE9C | ERRERICR2 |

Accesses on this interface are **RW**.

ERRFHICR0, Fault Handling Interrupt Configuration Register 0

The ERRFHICR0 characteristics are:

Purpose

Fault Handling Interrupt configuration register.

Configuration

This register is present only when (the Fault Handling Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR<n> registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRFHICR0 are RES0.

ERRFHICR0 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRFHICR0 is a 64-bit register.

Field descriptions

When the Fault Handling Interrupt is implemented and the implementation uses the recommended layout for the ERRIRQCR<n> registers:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | | | | | | | | ADDR | | | | | | | | | | | | | | | | | | | | | | | | |
| ADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | RES0 | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bits [63:56]

Reserved, RES0.

ADDR, bits [55:2]

Message Signaled Interrupt address. (ERRFHICR0.ADDR << 2) is the address that the component writes to when signaling the Fault Handling Interrupt. Bits [1:0] of the address are always zero.

The physical address size supported by the component is IMPLEMENTATION DEFINED. Unimplemented high-order physical address bits are RES0.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Bits [1:0]

Reserved, RES0.

When the implementation does not use the recommended layout for the ERRIRQCR<n> registers:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRFHICR0

ERRFHICR0 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-----------|
| RAS | 0xE80 | ERRFHICR0 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRFHICR1, Fault Handling Interrupt Configuration Register 1

The ERRFHICR1 characteristics are:

Purpose

Fault Handling Interrupt configuration register.

Configuration

This register is present only when (the Fault Handling Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR<n> registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRFHICR1 are RES0.

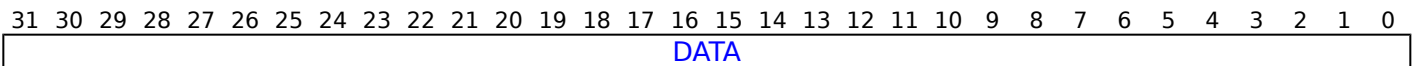
ERRFHICR1 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRFHICR1 is a 32-bit register.

Field descriptions

When the Fault Handling Interrupt is implemented and the implementation uses the recommended layout for the ERRIRQCR<n> registers:



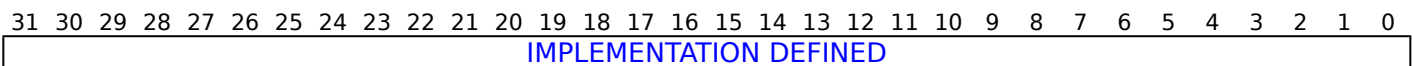
DATA, bits [31:0]

Payload for the message signaled interrupt.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

When the implementation does not use the recommended layout for the ERRIRQCR<n> registers:



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRFHICR1

ERRFHICR1 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-----------|
| RAS | 0xE88 | ERRFHICR1 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRFHICR2, Fault Handling Interrupt Configuration Register 2

The ERRFHICR2 characteristics are:

Purpose

Fault Handling Interrupt control and configuration register.

Configuration

This register is present only when (the Fault Handling Interrupt is implemented or the implementation does not use the recommended layout for the ERRIRQCR<n> registers) and interrupt configuration registers are implemented. Otherwise, direct accesses to ERRFHICR2 are RES0.

ERRFHICR2 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRFHICR2 is a 32-bit register.

Field descriptions

When the Fault Handling Interrupt is implemented and the implementation uses the recommended layout for the ERRIRQCR<n> registers:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|-------|----|-------|----|----|----|---------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | IRQEN | | NSMSI | | SH | | MemAttr | | | | | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

IRQEN, bit [7]

When the component supports disabling message signaled interrupts:

Message signaled interrupt enable. Enables generation of message signaled interrupts.

| IRQEN | Meaning |
|-------|-----------|
| 0b0 | Disabled. |
| 0b1 | Enabled. |

The reset behavior of this field is:

- On an Error recovery reset, this field resets to 0.

Otherwise:

Reserved, RES0.

Message signaled interrupt enable.

Message signaled interrupts are always enabled.

NSMSI, bit [6]

When the component supports configuring the Security attribute for message signaled interrupts and the component does not allow Non-secure writes to ERRFHICR2:

Security attribute. Defines the physical address space for message signaled interrupts.

| NSMSI | Meaning |
|-------|-------------|
| 0b0 | Secure. |
| 0b1 | Non-secure. |

The reset behavior of this field is:

- On a Error recovery reset, this field resets to an IMPLEMENTATION DEFINED value.

When the component allows Non-secure writes to ERRFHICR2:

Reserved, RES0.

Security attribute. Defines the physical address space for message signaled interrupts.

The Security attribute used for message signaled interrupts is Non-secure.

Otherwise:

Reserved, RES0.

Security attribute. Defines the physical address space for message signaled interrupts.

The Security attribute for message signaled interrupts is IMPLEMENTATION DEFINED.

SH, bits [5:4]

When the component supports configuring the Shareability domain for message signaled interrupts:

Shareability. Defines the Shareability domain for message signaled interrupts.

| SH | Meaning |
|------|------------------|
| 0b00 | Not shared. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

All other values are reserved.

This field is ignored when ERRFHICR2.MemAttr specifies any of the following memory types:

- Any Device memory type.
- Normal memory, Inner Non-cacheable, Outer Non-cacheable.

All Device and Normal Inner Non-cacheable Outer Non-cacheable memory regions are always treated as Outer Shareable.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Shareability.

The Shareability domain for message signaled interrupts is IMPLEMENTATION DEFINED.

MemAttr, bits [3:0]

When the component supports configuring the memory type for message signaled interrupts:

Memory type. Defines the memory type and attributes for message signaled interrupts.

| MemAttr | Meaning |
|---------|--|
| 0b0000 | Device-nGnRnE memory. |
| 0b0001 | Device-nGnRE memory. |
| 0b0010 | Device-nGRE memory. |
| 0b0011 | Device-GRE memory. |
| 0b0101 | Normal memory, Inner Non-cacheable, Outer Non-cacheable. |
| 0b0110 | Normal memory, Inner Write-Through, Outer Non-cacheable. |
| 0b0111 | Normal memory, Inner Write-Back, Outer Non-cacheable. |
| 0b1001 | Normal memory, Inner Non-cacheable, Outer Write-Through. |
| 0b1010 | Normal memory, Inner Write-Through, Outer Write-Through. |
| 0b1011 | Normal memory, Inner Write-Back, Outer Write-Through. |
| 0b1101 | Normal memory, Inner Non-cacheable, Outer Write-Back. |
| 0b1110 | Normal memory, Inner Write-Through, Outer Write-Back. |
| 0b1111 | Normal memory, Inner Write-Back, Outer Write-Back. |

All other values are reserved.

Note

This is the same format as the VMSAv8-64 stage 2 memory region attributes.

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Memory type.

The memory type used for message signaled interrupts is IMPLEMENTATION DEFINED.

When the implementation does not use the recommended layout for the ERRIRQCR<n> registers:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

Accessing ERRFHICR2

ERRFHICR2 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|-----------|
| RAS | 0xE8C | ERRFHICR2 |

Accesses on this interface are **RW**.

ERRGSR, Error Group Status Register

The ERRGSR characteristics are:

Purpose

Shows the status for the records in the group.

Configuration

ERRGSR is implemented only as part of a memory-mapped group of error records.

This manual describes a group of error records accessed via a standard 4KB memory-mapped peripheral. For a 4KB peripheral, up to 24 error records can be accessed if the Common Fault Injection Model is implemented, and up to 56 otherwise.

Attributes

ERRGSR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 |
| RES0 | | | | | | | | S55 | S54 | S53 | S52 | S51 | S50 | S49 | S48 | S47 | S46 | S45 | S44 | S43 | S42 | S41 | S40 | S39 | S38 | S37 | S36 | S35 |
| S31 | S30 | S29 | S28 | S27 | S26 | S25 | S24 | S23 | S22 | S21 | S20 | S19 | S18 | S17 | S16 | S15 | S14 | S13 | S12 | S11 | S10 | S9 | S8 | S7 | S6 | S5 | S4 | S3 |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 |

Bits [63:56]

Reserved, RES0.

S<m>, bit [m], for m = 55 to 0

When error record <m> is implemented and error record <m> supports this type of reporting:

The status for error record <m>. A read-only copy of [ERR<m>STATUS.V](#).

| S<m> | Meaning |
|------|---------------------|
| 0b0 | No error. |
| 0b1 | One or more errors. |

If the Common Fault Injection Model is implemented, up-to 24 records can be implemented meaning bits [55:24] are RES0.

Otherwise:

Reserved, RES0.

Accessing ERRGSR

ERRGSR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| RAS | 0xE00 | ERRGSR |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRIIDR, Implementation Identification Register

The ERRIIDR characteristics are:

Purpose

Defines the implementer of the component.

Configuration

Implementation of this register is OPTIONAL.

This register is present only when RAS System Architecture v1.1 is implemented.

Attributes

ERRIIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----------|----|----|----|-------------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ProductID | | | | | | | | | | | | Variant | | | | Revision | | | | Implementer | | | | | | | | | | | |

ProductID, bits [31:20]

Part number, bits [11:0]. The part number is selected by the designer of the component.

If [ERRPIDR0](#) and [ERRPIDR1](#) are implemented, [ERRPIDR0](#).PART_0 matches bits [7:0] of ERRIIDR.ProductID and [ERRPIDR1](#).PART_1 matches bits [11:8] of ERRIIDR.ProductID.

Variant, bits [19:16]

Component major revision.

This field distinguishes product variants or major revisions of the product.

If [ERRPIDR2](#) is implemented, [ERRPIDR2](#).REVISION matches ERRIIDR.Variant.

Revision, bits [15:12]

Component minor revision.

This field distinguishes minor revisions of the product.

If [ERRPIDR3](#) is implemented, [ERRPIDR3](#).REVAND matches ERRIIDR.Revision.

Implementer, bits [11:0]

Contains the JEP106 code of the company that implemented the RAS component. For an Arm implementation, this field has the value 0x43B.

Bits [11:8] contain the JEP106 continuation code of the implementer, and bits [6:0] contain the JEP106 identity code of the implementer. Bit 7 is RES0.

If [ERRPIDR4](#) is implemented, [ERRPIDR2](#) is implemented, and [ERRPIDR1](#) is implemented, [ERRPIDR4](#).DES_2 matches bits [11:8] of ERRIIDR.Implementer, [ERRPIDR2](#).DES_1 matches bits [6:4] of ERRIIDR.Implementer, and [ERRPIDR1](#).DES_0 matches bits [3:0] of ERRIIDR.Implementer.

Accessing ERRIIDR

ERRIIDR can be accessed through the memory-mapped interfaces:

| Component | Offset |
|-----------|--------|
| RAS | 0xE10 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRIMPDEF<n>, IMPLEMENTATION DEFINED Register <n>, n = 0 - 191

The ERRIMPDEF<n> characteristics are:

Purpose

IMPLEMENTATION DEFINED RAS extensions.

Configuration

This register is present only when the Common Fault Injection Model Extension is not implemented, ERRDEVID.NUM <= 32 and an implementation implements ERRIMPDEF<n>. Otherwise, direct accesses to ERRIMPDEF<n> are RES0.

Attributes

ERRIMPDEF<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRIMPDEF<n>

ERRIMPDEF<n> can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|-----------------|--------------|
| RAS | 0x800 + (8 * n) | ERRIMPDEF<n> |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRIRQCR<n>, Generic Error Interrupt Configuration Register, n = 0 - 15

The ERRIRQCR<n> characteristics are:

Purpose

The ERRIRQCR<n> registers are reserved for IMPLEMENTATION DEFINED interrupt configuration registers.

The architecture provides a recommended layout for the ERRIRQCR<n> registers. These registers are named:

- [ERRFHICR0](#), [ERRFHICR1](#), and [ERRFHICR2](#) for the fault handling interrupt controls.
- [ERRERICR0](#), [ERRERICR1](#), and [ERRERICR2](#) for the error recovery interrupt controls.
- [ERRCRICR0](#), [ERRCRICR1](#), and [ERRCRICR2](#) for the critical error interrupt controls.
- [ERRIRQSR](#) for the status register.

This section describes the generic, IMPLEMENTATION DEFINED, format.

Configuration

This register is present only when the interrupt configuration registers are implemented. Otherwise, direct accesses to ERRIRQCR<n> are RES0.

ERRIRQCR<n> is implemented only as part of a memory-mapped group of error records.

Attributes

ERRIRQCR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED controls. The content of these registers is IMPLEMENTATION DEFINED.

Accessing ERRIRQCR<n>

ERRIRQCR<n> can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|-----------------|-------------|
| RAS | 0xE80 + (8 * n) | ERRIRQCR<n> |

Accesses on this interface are **RW**.

ERRIRQSR, Error Interrupt Status Register

The ERRIRQSR characteristics are:

Purpose

Interrupt status register.

Configuration

This register is present only when interrupt configuration registers are implemented. Otherwise, direct accesses to ERRIRQSR are RES0.

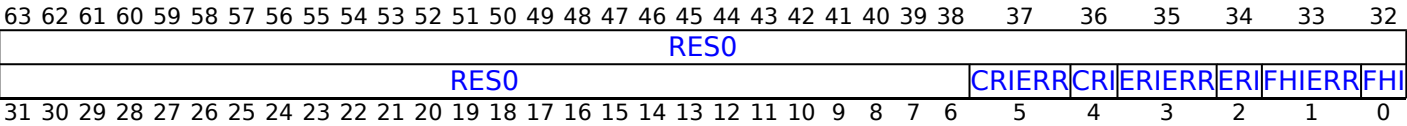
ERRIRQSR is implemented only as part of a memory-mapped group of error records.

Attributes

ERRIRQSR is a 64-bit register.

Field descriptions

When the implementation uses the recommended layout for the ERRIRQCR<n> registers:



Bits [63:6]

Reserved, RES0.

CRIERR, bit [5]

When the Critical Error Interrupt is implemented:

Critical Error Interrupt Error.

| CRIERR | Meaning |
|--------|---|
| 0b0 | Critical Error Interrupt write has not returned an error since this field was last cleared to zero. |
| 0b1 | Critical Error Interrupt write has returned an error since this field was last cleared to zero. |

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

CRI, bit [4]**When the Critical Error Interrupt is implemented:**

Critical Error Interrupt write in progress.

| CRI | Meaning |
|------------|---|
| 0b0 | Critical Error Interrupt write not in progress. |
| 0b1 | Critical Error Interrupt write in progress. |

Software must not disable an interrupt whilst the write is in progress.

Note

This field does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

To determine whether an interrupt is active, software must examine the individual [ERR<n>STATUS](#) registers.

Access to this field is **RO**.**Otherwise:**

Reserved, RES0.

ERIERR, bit [3]**When the Error Recovery Interrupt is implemented:**

Error Recovery Interrupt Error.

| ERIERR | Meaning |
|---------------|---|
| 0b0 | Error Recovery Interrupt write has not returned an error since this field was last cleared to zero. |
| 0b1 | Error Recovery Interrupt write has returned an error since this field was last cleared to zero. |

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **W1C**.**Otherwise:**

Reserved, RES0.

ERI, bit [2]**When the Error Recovery Interrupt is implemented:**

Error Recovery Interrupt write in progress.

| ERI | Meaning |
|------------|---|
| 0b0 | Error Recovery Interrupt write not in progress. |
| 0b1 | Error Recovery Interrupt write in progress. |

Software must not disable an interrupt whilst the write is in progress.

Note

This field does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

To determine whether an interrupt is active, software must examine the individual [ERR<n>STATUS](#) registers.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

FHIERR, bit [1]

When the Fault Handling Interrupt is implemented:

Fault Handling Interrupt Error.

| FHIERR | Meaning |
|---------------|---|
| 0b0 | Fault Handling Interrupt write has not returned an error since this field was last cleared to zero. |
| 0b1 | Fault Handling Interrupt write has returned an error since this field was last cleared to zero. |

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

FHI, bit [0]

When the Fault Handling Interrupt is implemented:

Fault Handling Interrupt write in progress.

| FHI | Meaning |
|------------|---|
| 0b0 | Fault Handling Interrupt write not in progress. |
| 0b1 | Fault Handling Interrupt write in progress. |

Software must not disable an interrupt whilst the write is in progress.

Note

This field does not indicate whether an interrupt is active, but rather whether a write triggered by the interrupt is in progress.

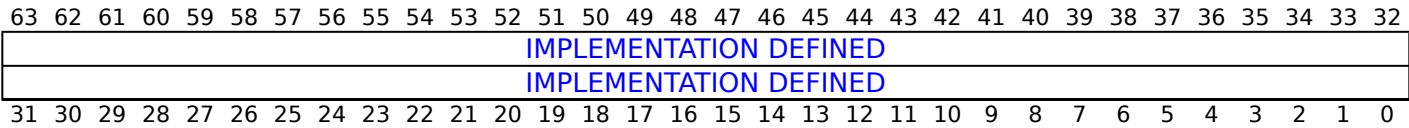
To determine whether an interrupt is active, software must examine the individual [ERR<n>STATUS](#) registers.

Access to this field is **RO**.

Otherwise:

Reserved, RES0.

When the implementation does not use the recommended layout for the ERRIRQCR<n> registers:



IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing ERRIRQSR

ERRIRQSR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| RAS | 0xEF8 | ERRIRQSR |

Accesses on this interface are **RW**.

ERR<n>ADDR, Error Record Address Register, n = 0 - 65534

The ERR<n>ADDR characteristics are:

Purpose

If an address is associated with a detected error, then it is written to ERR<n>ADDR when the error is recorded. It is IMPLEMENTATION DEFINED how the recorded address maps to the software-visible physical address. Software might have to reconstruct the actual physical addresses using the identity of the node and knowledge of the system.

Configuration

This register is present only when error record <n> is implemented and error record <n> includes an address associated with an error. Otherwise, direct accesses to ERR<n>ADDR are RES0.

[ERR<q>FR](#) describes the features implemented by the node that owns error record <n>. <q> is the index of the first error record owned by the same node as error record <n>. If the node owns a single record, then q = n.

Attributes

ERR<n>ADDR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|-----|------|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| NS | SI | AI | VA | NSE | RES0 | | PADDR | | | | | | | | | | | | | | | | | | | | | | | | |
| PADDR | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

NS, bit [63]

When FEAT_RME is implemented:

Non-secure attribute. With ERR<n>ADDR.NSE, indicates the physical address space of the recorded location.

| NS | Meaning |
|-----|---|
| 0b0 | When ERR<n>ADDR.NSE == 0: ERR<n>ADDR.PADDR is a Secure address. When ERR<n>ADDR.NSE == 1: ERR<n>ADDR.PADDR is a Root address. |
| 0b1 | When ERR<n>ADDR.NSE == 0: ERR<n>ADDR.PADDR is a Non-secure address. When ERR<n>ADDR.NSE == 1: ERR<n>ADDR.PADDR is a Realm address. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When FEAT_RME is not implemented:

Non-secure attribute.

| NS | Meaning |
|-----|---|
| 0b0 | ERR<n>ADDR.PADDR is a Secure address. |
| 0b1 | ERR<n>ADDR.PADDR is a Non-secure address. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SI, bit [62]

When FEAT_RME is implemented:

Secure Incorrect. Indicates whether ERR<n>ADDR.{NS, NSE} are valid.

| SI | Meaning |
|-----|---|
| 0b0 | ERR<n>ADDR.{NS, NSE} are correct. That is, they match the programmers' view of the physical address space for the recorded location. |
| 0b1 | ERR<n>ADDR.{NS, NSE} might not be correct, and might not match the programmers' view of the physical address space for the recorded location. |

It is IMPLEMENTATION DEFINED whether this field is read-only or read/write.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When FEAT_RME is not implemented:

Secure Incorrect. Indicates whether ERR<n>ADDR.NS is valid.

| SI | Meaning |
|-----|--|
| 0b0 | ERR<n>ADDR.NS is correct. That is, it matches the programmers' view of the Non-secure attribute for the recorded location. |
| 0b1 | ERR<n>ADDR.NS might not be correct, and might not match the programmers' view of the Non-secure attribute for the recorded location. |

It is IMPLEMENTATION DEFINED whether this field is read-only or read/write.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

AI, bit [61]

Address Incorrect. Indicates whether ERR<n>ADDR.PADDR is a valid physical address that is known to match the programmers' view of the physical address for the recorded location.

| AI | Meaning |
|-----|--|
| 0b0 | ERR<n>ADDR.PADDR is a valid physical address. That is, it matches the programmers' view of the physical address for the recorded location. |
| 0b1 | ERR<n>ADDR.PADDR might not be a valid physical address, and might not match the programmers' view of the physical address for the recorded location. |

It is IMPLEMENTATION DEFINED whether this field is read-only or read/write.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

VA, bit [60]

Virtual Address. Indicates whether ERR<n>ADDR.PADDR field is a virtual address.

| VA | Meaning |
|-----|--|
| 0b0 | ERR<n>ADDR.PADDR is not a virtual address. |
| 0b1 | ERR<n>ADDR.PADDR is a virtual address. |

No context information is provided for the virtual address. When ERR<n>ADDR.VA is 1, ERR<n>ADDR.{NS, SI, AI} read as {0, 1, 1}.

Support for this field is optional. If this field is not implemented and ERR<n>ADDR.PADDR field is a virtual address, then ERR<n>ADDR.{NS, SI, AI} read as {0, 1, 1}.

It is IMPLEMENTATION DEFINED whether this field is read-only or read/write.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

NSE, bit [59]

When FEAT_RME is implemented:

Physical Address Space. Together with ERR<n>ADDR.NS, indicates the address space for ERR<n>ADDR.PADDR.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [58:56]

Reserved, RES0.

PADDR, bits [55:0]

Physical Address. Address of the recorded location. If the physical address size implemented by this component is smaller than the size of this field, then high-order bits are unimplemented and either RES0 or have a fixed read-only IMPLEMENTATION DEFINED value. Low-order address bits might also be unimplemented and RES0, for example, if the physical address is always aligned to the size of a protection granule.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing ERR<n>ADDR

ERR<n>ADDR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|------------------|------------|
| RAS | 0x018 + (64 * n) | ERR<n>ADDR |

This interface is accessible as follows:

- When the Common Fault Injection Model Extension is implemented by the node that owns this error record, ERR<q>PFGF.AV == 0 and ERR<n>STATUS.AV == 1 accesses to this register are **RO**.
- When the Common Fault Injection Model Extension is not implemented by the node that owns this error record and ERR<n>STATUS.AV == 1 accesses to this register are **RO**.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>CTLR, Error Record Control Register, n = 0 - 65534

The ERR<n>CTLR characteristics are:

Purpose

The error control register contains enable bits for the node that writes to this record:

- Enabling error detection and correction.
- Enabling the critical error, error recovery, and fault handling interrupts.
- Enabling in-band error response for uncorrected errors.

For each bit, if the node does not support the feature, then the bit is RES0. The definition of each record is IMPLEMENTATION DEFINED.

Configuration

This register is present only when error record <n> is implemented and error record <n> is the first error record owned by a node. Otherwise, direct accesses to ERR<n>CTLR are RES0.

[ERR<n>FR](#) describes the features implemented by the node.

Attributes

ERR<n>CTLR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|------|---------|----|----|--------|-----|-----|-----|--------|--------|--------|------------------------|--|--|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | CI | RES0 | WDUI | Bit[10] | WC | FI | Bit[8] | WUE | WFI | WUI | Bit[4] | Bit[3] | Bit[2] | IMPLEMENTATION DEFINED | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | | | | | | | |

IMPLEMENTATION DEFINED, bits [63:32]

Reserved for IMPLEMENTATION DEFINED controls. Must permit SBZP write policy for software.

Bits [31:14]

Reserved, RES0.

CI, bit [13]

When ERR<n>FR.CI == 0b10:

Critical error interrupt enable. When enabled, the critical error interrupt is generated for a critical error condition.

| CI | Meaning |
|-----|--|
| 0b0 | Critical error interrupt not generated for critical errors. Critical errors are treated as Uncontained errors. |
| 0b1 | Critical error interrupt generated for critical errors. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [12]

Reserved, RES0.

WDUI, bit [11]

When **ERR<n>FR.DUI == 0b11:**

Error recovery interrupt for Deferred errors on writes enable.

When enabled, the error recovery interrupt is generated for errors recorded as Deferred error on writes.

| WDUI | Meaning |
|------|---|
| 0b0 | Error recovery interrupt not generated for Deferred errors on writes. |
| 0b1 | Error recovery interrupt generated for Deferred errors on writes. |

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DUI, bit [10]

When **ERR<n>FR.DUI == 0b10:**

Error recovery interrupt for Deferred errors enable.

When [ERR<n>FR.DUI == 0b10](#), this control applies to errors arising from both reads and writes.

When enabled, the error recovery interrupt is generated for all errors recorded as Deferred error.

| DUI | Meaning |
|-----|---|
| 0b0 | Error recovery interrupt not generated for Deferred errors. |
| 0b1 | Error recovery interrupt generated for Deferred errors. |

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When **ERR<n>FR.DUI == 0b11:**

Error recovery interrupt for Deferred errors on reads enable.

When [ERR<n>FR.DUI == 0b11](#), this field is named RDUI.

When enabled, the error recovery interrupt is generated for errors recorded as Deferred error on reads.

| RDUI | Meaning |
|-------------|--|
| 0b0 | Error recovery interrupt not generated for Deferred errors on reads. |
| 0b1 | Error recovery interrupt generated for Deferred errors on reads. |

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WCFI, bit [9]

When ERR<n>FR.CFI == 0b11:

Fault handling interrupt for Corrected errors on writes enable.

When enabled:

- If the node implements Corrected error counters for writes, then the fault handling interrupt is generated when a counter overflows and the overflow bit for the counter is set to 1. For more information, see [ERR<n>MISC0](#).
- Otherwise, the fault handling interrupt is also generated for errors recorded as Corrected error on writes.

| WCFI | Meaning |
|-------------|--|
| 0b0 | Fault handling interrupt not generated for Corrected errors on writes. |
| 0b1 | Fault handling interrupt generated for Corrected errors on writes. |

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CFI, bit [8]

When ERR<n>FR.CFI == 0b10:

Fault handling interrupt for Corrected errors enable.

When [ERR<n>FR.CFI](#) == 0b10, this control applies to errors arising from both reads and writes.

When enabled:

- If the node implements Corrected error counters, then the fault handling interrupt is generated when a counter overflows and the overflow bit for the counter is set to 1. For more information, see [ERR<n>MISC0](#).
- Otherwise, the fault handling interrupt is also generated for all errors recorded as Corrected error.

| CFI | Meaning |
|------------|--|
| 0b0 | Fault handling interrupt not generated for Corrected errors. |
| 0b1 | Fault handling interrupt generated for Corrected errors. |

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ERR<n>FR.CFI == 0b11:

Fault handling interrupt for Corrected errors on reads enable.

When [ERR<n>FR.CFI](#) == 0b11, this field is named RCFI.

When enabled:

- If the node implements Corrected error counters for reads, then the fault handling interrupt is generated when a counter overflows and the overflow bit for the counter is set to 1. For more information, see [ERR<n>MISC0](#).
- Otherwise, the fault handling interrupt is also generated for errors recorded as Corrected error on reads.

| RCFI | Meaning |
|------|---|
| 0b0 | Fault handling interrupt not generated for Corrected errors on reads. |
| 0b1 | Fault handling interrupt generated for Corrected errors on reads. |

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WUE, bit [7]

When ERR<n>FR.UE == 0b11:

In-band error response on writes enable.

When enabled, responses to writes that detect an error that is not corrected and is not deferred are signaled with an in-band error response (External Abort).

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester.

| WUE | Meaning |
|-----|---|
| 0b0 | In-band error response for uncorrected errors on writes disabled. |
| 0b1 | In-band error response for uncorrected errors on writes enabled. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WFI, bit [6]

When ERR<n>FR.FI == 0b11:

Fault handling interrupt on writes enable.

When enabled:

- The fault handling interrupt is generated for errors recorded as either Deferred error or Uncorrected error on writes.
- If the corresponding fault handling interrupt for Corrected errors control is not implemented:
 - If the node implements Corrected error counters for writes, then the fault handling interrupt is also generated when a counter overflows and the overflow bit for the counter is set to 1.
 - Otherwise, the fault handling interrupt is also generated for errors recorded as Corrected error on writes.

| WFI | Meaning |
|-----|--|
| 0b0 | Fault handling interrupt on writes disabled. |
| 0b1 | Fault handling interrupt on writes enabled. |

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

WUI, bit [5]

When **ERR<n>FR.UI == 0b11**:

Uncorrected error recovery interrupt on writes enable.

When enabled, the error recovery interrupt is generated for errors recorded as Uncorrected error on writes.

| WUI | Meaning |
|-----|--|
| 0b0 | Error recovery interrupt on writes disabled. |
| 0b1 | Error recovery interrupt on writes enabled. |

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UE, bit [4]

When **ERR<n>FR.UE == 0b10**:

In-band error response enable.

When [ERR<n>FR.UE == 0b10](#), this control applies to errors arising from both reads and writes.

When enabled, responses to transactions that detect an error that is not corrected and is not deferred are signaled with an in-band error response (External Abort).

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester.

| UE | Meaning |
|-----|---|
| 0b0 | In-band error response for uncorrected errors disabled. |
| 0b1 | In-band error response for uncorrected errors enabled. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ERR<n>FR.UE == 0b11:

In-band error response on reads enable.

When [ERR<n>FR.UE](#) == 0b11, this field is named RUE.

When enabled, responses to reads that detect an error that is not corrected and is not deferred are signaled with an in-band error response (External Abort).

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester.

| RUE | Meaning |
|-----|--|
| 0b0 | In-band error response for uncorrected errors on reads disabled. |
| 0b1 | In-band error response for uncorrected errors on reads enabled. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

FI, bit [3]**When ERR<n>FR.FI == 0b10:**

Fault handling interrupt enable.

When [ERR<n>FR.FI](#) == 0b10, this control applies to errors arising from both reads and writes.

When enabled:

- The fault handling interrupt is generated for all errors recorded as either Deferred error or Uncorrected error.
- If the fault handling interrupt for Corrected errors control is not implemented:
 - If the node implements Corrected error counters, then the fault handling interrupt is also generated when a counter overflows and the overflow bit for the counter is set to 1.
 - Otherwise, the fault handling interrupt is also generated for all errors recorded as Corrected error.

| FI | Meaning |
|-----|------------------------------------|
| 0b0 | Fault handling interrupt disabled. |
| 0b1 | Fault handling interrupt enabled. |

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When ERR<n>FR.FI == 0b11:

Fault handling interrupt on reads enable.

When [ERR<n>FR.FI](#) == 0b11, this field is named RFI.

When enabled:

- The fault handling interrupt is generated for errors recorded as either Deferred error or Uncorrected error on reads.
- If the corresponding fault handling interrupt for Corrected errors control is not implemented:
 - If the node implements Corrected error counters for reads, then the fault handling interrupt is also generated when a counter overflows and the overflow bit for the counter is set to 1.
 - Otherwise, the fault handling interrupt is also generated for errors recorded as Corrected error on reads.

| RFI | Meaning |
|-----|---|
| 0b0 | Fault handling interrupt on reads disabled. |
| 0b1 | Fault handling interrupt on reads enabled. |

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UI, bit [2]

When **ERR<n>FR.UI == 0b10:**

Uncorrected error recovery interrupt enable.

When [ERR<n>FR.UI == 0b10](#), this control applies to errors arising from both reads and writes.

When enabled, the error recovery interrupt is generated for all errors recorded as Uncorrected error.

| UI | Meaning |
|-----|------------------------------------|
| 0b0 | Error recovery interrupt disabled. |
| 0b1 | Error recovery interrupt enabled. |

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

When **ERR<n>FR.UI == 0b11:**

Uncorrected error recovery interrupt on reads enable.

When [ERR<n>FR.UI == 0b11](#), this field is named RUI.

When enabled, the error recovery interrupt is generated for errors recorded as Uncorrected error on reads.

| RUI | Meaning |
|-----|---|
| 0b0 | Error recovery interrupt on reads disabled. |
| 0b1 | Error recovery interrupt on reads enabled. |

The interrupt is generated even if the error syndrome is discarded because the error record already records a higher priority error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

IMPLEMENTATION DEFINED, bit [1]

Reserved for IMPLEMENTATION DEFINED controls. Must permit SBZP write policy for software.

ED, bit [0]

When ERR<n>FR.ED == 0b10:

Error reporting and logging enable. When disabled, the node behaves as if error detection and correction are disabled, and no errors are recorded or signaled by the node. Arm recommends that, when disabled, correct error detection and correction codes are written for writes, unless disabled by an IMPLEMENTATION DEFINED control for error injection.

| ED | Meaning |
|-----|---------------------------|
| 0b0 | Error reporting disabled. |
| 0b1 | Error reporting enabled. |

It is IMPLEMENTATION DEFINED whether the node fully disables error detection and correction when reporting is disabled. That is, even with error reporting disabled, the node might continue to silently correct errors. Uncorrected errors might result in corrupt data being silently propagated by the node.

Note

If this node requires initialization after Cold reset to prevent signaling false errors, then Arm recommends this field is set to 0 on Cold reset, meaning errors are not reported from Cold reset. This allows boot software to initialize a node without signaling errors. Software can enable error reporting after the node is initialized. Otherwise, the Cold reset value is IMPLEMENTATION DEFINED. If the Cold reset value is 1, the reset values of other controls in this register are also IMPLEMENTATION DEFINED and should not be UNKNOWN.

The reset behavior of this field is:

- On a Cold reset, this field resets to an IMPLEMENTATION DEFINED value.

Otherwise:

Reserved, RES0.

Accessing ERR<n>CTLR

ERR<n>CTLR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|------------------|------------|
| RAS | 0x008 + (64 * n) | ERR<n>CTLR |

Accesses on this interface are RW.

ERR<n>FR, Error Record Feature Register, n = 0 - 65534

The ERR<n>FR characteristics are:

Purpose

Defines whether <n> is the first record owned by a node:

- If <n> is the first error record owned by a node, then ERR<n>FR.ED is not 0b00.
- If <n> is not the first error record owned by a node, then ERR<n>FR.ED is 0b00.

If <n> is the first record owned by the node, defines which of the common architecturally-defined features are implemented by the node and, of the implemented features, which are software programmable.

Configuration

This register is present only when error record <n> is implemented. Otherwise, direct accesses to ERR<n>FR are RES0.

Attributes

ERR<n>FR is a 64-bit register.

Field descriptions

When error record <n> is not the first error record owned by the node:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | ED |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:2]

Reserved, RES0.

ED, bits [1:0]

Error reporting and logging. Indicates error record <n> is not the first error record owned the node.

| ED | Meaning |
|------|---|
| 0b00 | Error record <n> is not the first error record owned by the node. |

This field reads as 0b00.

When error record <n> is the first error record owned by the node:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------------|------|----|----|----|----|----|----|-----|-----|----|----|----|-----|----|-----|---------------------------|----|----|----|---------------------------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | CE | DE | UE | OU | ER | UE | UC | IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | |
| FRX | RES0 | | | | | TS | CI | INJ | CEO | | | | DUI | RP | CEC | CFI | UE | FI | UI | IMPLEMENTATION DEFINED | | | | | ED | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:55]**When ERR<n>FR.FRX != 1:**

Reserved for identifying IMPLEMENTATION DEFINED controls.

Otherwise:

Reserved, RES0.

CE, bits [54:53]**When ERR<n>FR.FRX == 1:**

Corrected Error recording. Describes the types of Corrected errors the node can record, if any.

| CE | Meaning |
|------|--|
| 0b00 | Does not record Corrected errors. |
| 0b01 | Records only transient or persistent Corrected errors. That is, Corrected errors recorded by setting ERR<n>STATUS.CE to either 0b01 or 0b11. |
| 0b10 | Records only non-specific Corrected errors. That is, Corrected errors recorded by setting ERR<n>STATUS.CE to 0b10. |
| 0b11 | Records all types of Corrected error. |

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

DE, bit [52]**When ERR<n>FR.FRX == 1:**

Deferred Error recording. Describes whether the node supports recording Deferred errors.

| DE | Meaning |
|-----|----------------------------------|
| 0b0 | Does not record Deferred errors. |
| 0b1 | Records Deferred errors. |

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

UEO, bit [51]**When ERR<n>FR.FRX == 1:**

Latent or Restartable Error recording. Describes whether the node supports recording Latent or Restartable errors.

| UEO | Meaning |
|-----|---|
| 0b0 | Does not record Latent or Restartable errors. |
| 0b1 | Records Latent or Restartable errors. |

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

UER, bit [50]**When ERR<n>FR.FRX == 1:**

Signaled or Recoverable Error recording. Describes whether the node supports recording Signaled or Recoverable errors.

| UER | Meaning |
|-----|---|
| 0b0 | Does not record Signaled or Recoverable errors. |
| 0b1 | Records Signaled or Recoverable errors. |

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

UEU, bit [49]**When ERR<n>FR.FRX == 1:**

Unrecoverable Error recording. Describes whether the node supports recording Unrecoverable errors.

| UEU | Meaning |
|-----|---------------------------------------|
| 0b0 | Does not record Unrecoverable errors. |
| 0b1 | Records Unrecoverable errors. |

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

UC, bit [48]**When ERR<n>FR.FRX == 1:**

Uncontainable Error recording. Describes whether the node supports recording Uncontainable errors.

| UC | Meaning |
|-----|---------------------------------------|
| 0b0 | Does not record Uncontainable errors. |
| 0b1 | Records Uncontainable errors. |

Otherwise:

Reserved for identifying IMPLEMENTATION DEFINED controls.

IMPLEMENTATION DEFINED, bits [47:32]

Reserved for identifying IMPLEMENTATION DEFINED controls.

FRX, bit [31]**When RAS System Architecture v1.1 is implemented:**

Feature Register extension. Defines whether ERR<n>FR[63:48] are architecturally defined.

| FRX | Meaning |
|-----|--|
| 0b0 | ERR<n>FR[63:48] are IMPLEMENTATION DEFINED. |
| 0b1 | ERR<n>FR[63:48] are defined by the architecture. |

Otherwise:

Reserved, RES0.

Bits [30:26]

Reserved, RES0.

TS, bits [25:24]

Timestamp Extension. Indicates whether, for each error record <m> owned by this node, [ERR<m>MISC3](#) is used as the timestamp register, and, if it is, the timebase used by the timestamp.

| TS | Meaning |
|---|---|
| 0b00 | Does not support a timestamp register. |
| 0b01 | Implements a timestamp register in ERR<n>MISC3 for each error record <m> owned by the node. The timestamp uses the same timebase as the system Generic Timer. |
| Note For an error record that has an affinity to a PE, this is the same timer that is visible through CNTPCT_ELO at the highest Exception level on that PE. | |
| 0b10 | Implements a timestamp register in ERR<m>MISC3 for each error record <m> owned by the node. The timestamp uses an IMPLEMENTATION DEFINED timebase. |

All other values are reserved.

CI, bits [23:22]

Critical error interrupt. Indicates whether the critical error interrupt and associated controls are implemented by the node.

| CI | Meaning |
|------|--|
| 0b00 | Does not support the critical error interrupt. ERR<n>CTLR.CI is RES0. |
| 0b01 | Critical error interrupt is supported and always enabled. ERR<n>CTLR.CI is RES0. |
| 0b10 | Critical error interrupt is supported and controllable using ERR<n>CTLR.CI . |

All other values are reserved.

INJ, bits [21:20]

Fault Injection Extension. Indicates whether the Common Fault Injection Model Extension is implemented by the node.

| INJ | Meaning |
|------|---|
| 0b00 | Does not support the Common Fault Injection Model Extension. |
| 0b01 | Supports the Common Fault Injection Model Extension. See ERR<n>PFGF for more information. |

All other values are reserved.

CEO, bits [19:18]**When ERR<n>FR.CEC != 0b000:**

Corrected Error overwrite. Indicates the behavior of the node when a second or subsequent Corrected error is recorded and a first Corrected error has previously been recorded by an error record <m> owned by the node.

| CEO | Meaning |
|------|---|
| 0b00 | Keeps the previous error syndrome. |
| 0b01 | If ERR<m>STATUS.OF is 1 before the Corrected error is counted, then the error record keeps the previous syndrome. Otherwise the previous syndrome is overwritten. |

All other values are reserved.

The second or subsequent Corrected error is counted by the Corrected error counter, regardless of the value of this field. If counting the error causes unsigned overflow of the counter, then [ERR<m>STATUS.OF](#) is set to 1.

This means that, if no other error is subsequently recorded that overwrites the syndrome:

- If ERR<n>FR.CEO is 0b00, the error record holds the syndrome for the first recorded Corrected error.
- If ERR<n>FR.CEO is 0b01, the error record holds the syndrome for the most recently recorded Corrected error before the counter overflows.

Otherwise:

Reserved, RES0.

DUI, bits [17:16]

When ERR<n>FR.UI != 0b00:

Error recovery interrupt for deferred errors control. Indicates whether the enabling and disabling of error recovery interrupts on deferred errors is supported by the node.

| DUI | Meaning |
|------|---|
| 0b00 | Does not support the enabling and disabling of error recovery interrupts on deferred errors. ERR<n>CTLR.DUI is RES0. |
| 0b10 | Enabling and disabling of error recovery interrupts on deferred errors is supported and controllable using ERR<n>CTLR.DUI . |
| 0b11 | Enabling and disabling of error recovery interrupts on deferred errors is supported, and controllable using ERR<n>CTLR.WDUI for writes and ERR<n>CTLR.RDUI for reads. |

All other values are reserved.

Otherwise:

Reserved, RES0.

RP, bit [15]

When ERR<n>FR.CEC != 0b000:

Repeat counter. Indicates whether the node implements a second Corrected error counter in [ERR<m>MISC0](#) for each error record <m> owned by the node that can record countable errors.

| RP | Meaning |
|-----|---|
| 0b0 | Implements a single Corrected error counter in ERR<m>MISC0 for each error record <m> owned by the node that can record countable errors. |
| 0b1 | Implements a first (repeat) counter and a second (other) counter in ERR<m>MISC0 for each error record <m> owned by the node that can record countable errors. The repeat counter is the same size as the primary error counter. |

Otherwise:

Reserved, RES0.

CEC, bits [14:12]

Corrected Error Counter. Indicates whether the node implements the standard Corrected error counter mechanisms in [ERR<m>MISC0](#) for each error record <m> owned by the node that can record countable errors.

| CEC | Meaning |
|-------|--|
| 0b000 | Does not implement the standard Corrected error counter model. |
| 0b010 | Implements an 8-bit Corrected error counter in ERR<m>MISC0 [39:32] for each error record <m> owned by the node that can record countable errors. |
| 0b100 | Implements a 16-bit Corrected error counter in ERR<m>MISC0 [47:32] for each error record <m> owned by the node that can record countable errors. |

All other values are reserved.

Note

Implementations might include other error counter models, or might include the standard model and not indicate this in ERR<n>FR.

CFI, bits [11:10]

When ERR<n>FR.FI != 0b00:

Fault handling interrupt for corrected errors control. Indicates whether the enabling and disabling of fault handling interrupts on corrected errors is supported by the node.

| CFI | Meaning |
|------|--|
| 0b00 | Does not support the enabling and disabling of fault handling interrupts on corrected errors. ERR<n>CTLR .CFI is RES0. |
| 0b10 | Enabling and disabling of fault handling interrupts on corrected errors is supported and controllable using ERR<n>CTLR .CFI. |
| 0b11 | Enabling and disabling of fault handling interrupts on corrected errors is supported, and controllable using ERR<n>CTLR .WCFI for writes and ERR<n>CTLR .RCFI for reads. |

All other values are reserved.

Otherwise:

Reserved, RES0.

UE, bits [9:8]

In-band error response (External Abort). Indicates whether the in-band error response and associated controls are implemented by the node.

| UE | Meaning |
|------|---|
| 0b00 | Does not support the in-band error response. ERR<n>CTLR .UE is RES0. |
| 0b01 | In-band error response is supported and always enabled. ERR<n>CTLR .UE is RES0. |
| 0b10 | In-band error response is supported and controllable using ERR<n>CTLR .UE. |
| 0b11 | In-band error response is supported, and controllable using ERR<n>CTLR .WUE for writes and ERR<n>CTLR .RUE for reads. |

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester.

FI, bits [7:6]

Fault handling interrupt. Indicates whether the fault handling interrupt and associated controls are implemented by the node.

| FI | Meaning |
|------|--|
| 0b00 | Does not support the fault handling interrupt. ERR<n>CTLR .FI is RES0. |
| 0b01 | Fault handling interrupt is supported and always enabled. ERR<n>CTLR .FI is RES0. |
| 0b10 | Fault handling interrupt is supported and controllable using ERR<n>CTLR .FI. |
| 0b11 | Fault handling interrupt is supported, and controllable using ERR<n>CTLR .WFI for writes and ERR<n>CTLR .RFI for reads. |

UI, bits [5:4]

Error recovery interrupt for uncorrected errors. Indicates whether the error handling interrupt and associated controls are implemented by the node.

| UI | Meaning |
|------|--|
| 0b00 | Does not support the error handling interrupt. ERR<n>CTLR .UI is RES0. |
| 0b01 | Error handling interrupt is supported and always enabled. ERR<n>CTLR .UI is RES0. |
| 0b10 | Error handling interrupt is supported and controllable using ERR<n>CTLR .UI. |
| 0b11 | Error handling interrupt is supported, and controllable using ERR<n>CTLR .WUI for writes and ERR<n>CTLR .RUI for reads. |

IMPLEMENTATION DEFINED, bits [3:2]

IMPLEMENTATION DEFINED.

ED, bits [1:0]

Error reporting and logging. Indicates error record <n> is the first record owned the node, and whether the node implements the controls for enabling and disabling error reporting and logging.

| ED | Meaning |
|------|--|
| 0b01 | Error reporting and logging always enabled. ERR<n>CTLR .ED is RES0. |
| 0b10 | Error reporting and logging is controllable using ERR<n>CTLR .ED. |

All other values are reserved.

Accessing ERR<n>FR

ERR<n>FR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|------------------|----------|
| RAS | 0x000 + (64 * n) | ERR<n>FR |

Accesses on this interface are **RO**.

ERR<n>MISC0, Error Record Miscellaneous Register 0, n = 0 - 65534

The ERR<n>MISC0 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to locate where the error was detected.
- If the error was detected within a FRU, the identity of the FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

If the node that owns error record <n> implements architecturally-defined Corrected error counters ([ERR<q>FR.CEC](#) != 0b000), and error record <n> can record countable errors, then ERR<n>MISC0 implements the architecturally-defined Corrected error counter or counters.

Configuration

This register is present only when error record <n> is implemented. Otherwise, direct accesses to ERR<n>MISC0 are RES0.

[ERR<q>FR](#) describes the features implemented by the node that owns error record <n>. <q> is the index of the first error record owned by the same node as error record <n>. If the node owns a single record, then q = n.

For IMPLEMENTATION DEFINED fields in ERR<n>MISC0, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero, and ignore writes are compliant with this requirement.

Note

Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in [ERR<q>CTLR](#).

Attributes

ERR<n>MISC0 is a 64-bit register.

Field descriptions

When ERR<q>FR.CEC == 0b000:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED syndrome.

When ERR<q>FR.CEC == 0b100 and ERR<q>FR.RP == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | OF | CEC | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

IMPLEMENTATION DEFINED, bits [63:48]

IMPLEMENTATION DEFINED syndrome.

OF, bit [47]

Sticky overflow bit. Set to 1 when ERR<n>MISC0.CEC is incremented and wraps through zero.

| OF | Meaning |
|-----|-----------------------------|
| 0b0 | Counter has not overflowed. |
| 0b1 | Counter has overflowed. |

A direct write that modifies this field might indirectly set [ERR<n>STATUS.OF](#) to an UNKNOWN value and a direct write to [ERR<n>STATUS.OF](#) that clears it to zero might indirectly set this field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CEC, bits [46:32]

Corrected error count. Incremented for each Corrected error. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED syndrome.

When ERR<q>FR.CEC == 0b010 and ERR<q>FR.RP == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | OF | CEC | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

IMPLEMENTATION DEFINED, bits [63:40]

IMPLEMENTATION DEFINED syndrome.

OF, bit [39]

Sticky overflow bit. Set to 1 when ERR<n>MISC0.CEC is incremented and wraps through zero.

| OF | Meaning |
|-----|-----------------------------|
| 0b0 | Counter has not overflowed. |
| 0b1 | Counter has overflowed. |

A direct write that modifies this field might indirectly set [ERR<n>STATUS.OF](#) to an UNKNOWN value and a direct write to [ERR<n>STATUS.OF](#) that clears it to zero might indirectly set this field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CEC, bits [38:32]

Corrected error count. Incremented for each Corrected error. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are counted.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED syndrome.

When ERR<q>FR.CEC == 0b100 and ERR<q>FR.RP == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| OFO | CECO | | | | | | | | | | | | | | | OFR | CECR | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

OFO, bit [63]

Sticky overflow bit, other. Set to 1 when ERR<n>MISC0.CECO is incremented and wraps through zero.

| OFO | Meaning |
|-----|-----------------------------------|
| 0b0 | Other counter has not overflowed. |
| 0b1 | Other counter has overflowed. |

A direct write that modifies this field might indirectly set [ERR<n>STATUS.OF](#) to an UNKNOWN value and a direct write to [ERR<n>STATUS.OF](#) that clears it to zero might indirectly set this field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CECO, bits [62:48]

Corrected error count, other. Incremented for each countable error that is not accounted for by incrementing ERR<n>MISC0.CECR.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

OFR, bit [47]

Sticky overflow bit, repeat. Set to 1 when ERR<n>MISC0.CECR is incremented and wraps through zero.

| OFR | Meaning |
|-----|------------------------------------|
| 0b0 | Repeat counter has not overflowed. |
| 0b1 | Repeat counter has overflowed. |

A direct write that modifies this field might indirectly set [ERR<n>STATUS.OF](#) to an UNKNOWN value and a direct write to [ERR<n>STATUS.OF](#) that clears it to zero might indirectly set this field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CECR, bits [46:32]

Corrected error count, repeat. Incremented for the first countable error, which also records other syndrome for the error, and subsequently for each countable error that matches the recorded other syndrome. Corrected errors are countable errors. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are countable errors.

Note

For example, the other syndrome might include the set and way information for an error detected in a cache. This might be recorded in the IMPLEMENTATION DEFINED ERR<n>MISC<m> fields on a first Corrected error. ERR<n>MISC0.CECR is then incremented for each subsequent Corrected Error in the same set and way.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED syndrome.

When ERR<q>FR.CEC == 0b010 and ERR<q>FR.RP == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|------|----|----|----|----|----|----|----|-----|------|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | OFO | CECO | | | | | | | | OFR | CECR | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

IMPLEMENTATION DEFINED, bits [63:48]

IMPLEMENTATION DEFINED syndrome.

OFO, bit [47]

Sticky overflow bit, other. Set to 1 when ERR<n>MISC0.CECO is incremented and wraps through zero.

| OFO | Meaning |
|-----|-----------------------------------|
| 0b0 | Other counter has not overflowed. |
| 0b1 | Other counter has overflowed. |

A direct write that modifies this field might indirectly set ERR<n>STATUS.OF to an UNKNOWN value and a direct write to ERR<n>STATUS.OF that clears it to zero might indirectly set this field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CECO, bits [46:40]

Corrected error count, other. Incremented for each countable error that is not accounted for by incrementing ERR<n>MISC0.CECR.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

OFR, bit [39]

Sticky overflow bit, repeat. Set to 1 when ERR<n>MISC0.CECR is incremented and wraps through zero.

| OFR | Meaning |
|-----|------------------------------------|
| 0b0 | Repeat counter has not overflowed. |
| 0b1 | Repeat counter has overflowed. |

A direct write that modifies this field might indirectly set [ERR<n>STATUS.OF](#) to an UNKNOWN value and a direct write to [ERR<n>STATUS.OF](#) that clears it to zero might indirectly set this field to an UNKNOWN value.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

CECR, bits [38:32]

Corrected error count, repeat. Incremented for the first countable error, which also records other syndrome for the error, and subsequently for each countable error that matches the recorded other syndrome. Corrected errors are countable errors. It is IMPLEMENTATION DEFINED and might be UNPREDICTABLE whether Deferred and Uncorrected errors are countable errors.

Note

For example, the other syndrome might include the set and way information for an error detected in a cache. This might be recorded in the IMPLEMENTATION DEFINED ERR<n>MISC<m> fields on a first Corrected error. ERR<n>MISC0.CECR is then incremented for each subsequent Corrected Error in the same set and way.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED syndrome.

Accessing ERR<n>MISC0

Reads from ERR<n>MISC0 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior.

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and [ERR<q>PFGF.MV](#) is 1, then some parts of this register are read/write when [ERR<n>STATUS.MV](#) is 1. See [ERR<n>PFGF.MV](#) for more information.

For other parts of this register, or if the Common Fault Injection Mechanism is not implemented, then Arm recommends that:

- Miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.
- When [ERR<n>STATUS.MV](#) is 1, the miscellaneous syndrome specific to the most recently recorded error ignores writes.

Note

These recommendations allow a counter to be reset in the presence of a persistent error, while preventing specific information, such as that identifying a FRU, from being lost if an error is detected while the previous error is being logged.

ERR<n>MISC0 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|------------------|-------------|
| RAS | 0x020 + (64 * n) | ERR<n>MISC0 |

Accesses on this interface are **RW**.

ERR<n>MISC1, Error Record Miscellaneous Register 1, n = 0 - 65534

The ERR<n>MISC1 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to locate where the error was detected.
- If the error was detected within a FRU, the identity of the FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

Configuration

This register is present only when error record <n> is implemented. Otherwise, direct accesses to ERR<n>MISC1 are RES0.

[ERR<q>FR](#) describes the features implemented by the node that owns error record <n>. <q> is the index of the first error record owned by the same node as error record <n>. If the node owns a single record, then q = n.

For IMPLEMENTATION DEFINED fields in ERR<n>MISC1, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero, and ignore writes are compliant with this requirement.

Note

Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in [ERR<q>CTLR](#).

Attributes

ERR<n>MISC1 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED syndrome.

Accessing ERR<n>MISC1

Reads from ERR<n>MISC1 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior.

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and [ERR<q>PFGF](#).MV is 1, then some parts of this register are read/write when [ERR<n>STATUS](#).MV is 1. See [ERR<n>PFGF](#).MV for more information.

For other parts of this register, or if the Common Fault Injection Mechanism is not implemented, then Arm recommends that:

- Miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.
- When [ERR<n>STATUS](#).MV is 1, the miscellaneous syndrome specific to the most recently recorded error ignores writes.

Note

These recommendations allow a counter to be reset in the presence of a persistent error, while preventing specific information, such as that identifying a FRU, from being lost if an error is detected while the previous error is being logged.

ERR<n>MISC1 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|------------------|-------------|
| RAS | 0x028 + (64 * n) | ERR<n>MISC1 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>MISC2, Error Record Miscellaneous Register 2, n = 0 - 65534

The ERR<n>MISC2 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to locate where the error was detected.
- If the error was detected within a FRU, the identity of the FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

Configuration

This register is present only when (an implementation implements ERR<n>MISC2 or RAS System Architecture v1.1 is implemented) and error record <n> is implemented. Otherwise, direct accesses to ERR<n>MISC2 are RES0.

[ERR<q>FR](#) describes the features implemented by the node that owns error record <n>. <q> is the index of the first error record owned by the same node as error record <n>. If the node owns a single record, then q = n.

For IMPLEMENTATION DEFINED fields in ERR<n>MISC2, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero, and ignore writes are compliant with this requirement.

If RAS System Architecture v1.1 is not implemented, Arm recommends that ERR<n>MISC2 does not require zeroing to return the record to a quiescent state.

Note

Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in [ERR<q>CTLR](#).

Attributes

ERR<n>MISC2 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED syndrome.

Accessing ERR<n>MISC2

Reads from ERR<n>MISC2 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior.

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and [ERR<q>PFGF](#).MV is 1, then some parts of this register are read/write when [ERR<n>STATUS](#).MV is 1. See [ERR<n>PFGF](#).MV for more information.

For other parts of this register, or if the Common Fault Injection Mechanism is not implemented, then Arm recommends that:

- Miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.
- When [ERR<n>STATUS](#).MV is 1, the miscellaneous syndrome specific to the most recently recorded error ignores writes.

Note

These recommendations allow a counter to be reset in the presence of a persistent error, while preventing specific information, such as that identifying a FRU, from being lost if an error is detected while the previous error is being logged.

ERR<n>MISC2 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|---------------------------|-------------|
| RAS | $0 \times 030 + (64 * n)$ | ERR<n>MISC2 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>MISC3, Error Record Miscellaneous Register 3, n = 0 - 65534

The ERR<n>MISC3 characteristics are:

Purpose

IMPLEMENTATION DEFINED error syndrome register. The miscellaneous syndrome registers might contain:

- Information to locate where the error was detected.
- If the error was detected within a FRU, the identity of the FRU.
- A Corrected error counter or counters.
- Other state information not present in the corresponding status and address registers.

If the node that owns error record n supports the RAS Timestamp Extension ([ERR<q>FR.TS](#) != 0b00), then ERR<n>MISC3 contains the timestamp value for error record n when the error was detected. Otherwise the contents of ERR<n>MISC3 are IMPLEMENTATION DEFINED.

Configuration

This register is present only when (an implementation implements ERR<n>MISC3 or RAS System Architecture v1.1 is implemented) and error record <n> is implemented. Otherwise, direct accesses to ERR<n>MISC3 are RES0.

[ERR<q>FR](#) describes the features implemented by the node that owns error record <n>. <q> is the index of the first error record owned by the same node as error record <n>. If the node owns a single record, then q = n.

For IMPLEMENTATION DEFINED fields in ERR<n>MISC3, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero, and ignore writes are compliant with this requirement.

If RAS System Architecture v1.1 is not implemented, Arm recommends that ERR<n>MISC3 does not require zeroing to return the record to a quiescent state.

Note

Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in [ERR<q>CTLR](#).

Attributes

ERR<n>MISC3 is a 64-bit register.

Field descriptions

When ERR<q>FR.TS != 0b00:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | TS | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | TS | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

TS, bits [63:0]

Timestamp. Timestamp value recorded when the error was detected. Valid only if [ERR<n>STATUS.V](#) == 1.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Access to this field is **RO** or **RW**.

When ERR<q>FR.TS == 0b00:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED syndrome.

Accessing ERR<n>MISC3

Reads from ERR<n>MISC3 return an IMPLEMENTATION DEFINED value and writes have IMPLEMENTATION DEFINED behavior.

If the Common Fault Injection Mechanism is implemented by the node that owns this error record, and [ERR<q>PFGF.MV](#) is 1, then some parts of this register are read/write when [ERR<n>STATUS.MV](#) is 1. See [ERR<n>PFGF.MV](#) for more information.

For other parts of this register, or if the Common Fault Injection Mechanism is not implemented, then Arm recommends that:

- Miscellaneous syndrome for multiple errors, such as a corrected error counter, is read/write.
- When [ERR<n>STATUS.MV](#) is 1, the miscellaneous syndrome specific to the most recently recorded error ignores writes.

Note

These recommendations allow a counter to be reset in the presence of a persistent error, while preventing specific information, such as that identifying a FRU, from being lost if an error is detected while the previous error is being logged.

ERR<n>MISC3 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|------------------|-------------|
| RAS | 0x038 + (64 * n) | ERR<n>MISC3 |

Accesses on this interface are **RW**.

ERR<n>PFGCDN, Pseudo-fault Generation Countdown Register, n = 0 - 65534

The ERR<n>PFGCDN characteristics are:

Purpose

Generates one of the errors enabled in the corresponding [ERR<n>PFGCTL](#) register.

Configuration

This register is present only when error record <n> is implemented, the node implements the Common Fault Injection Model Extension (ERR<n>FR.INJ != 0b00) and error record <n> is the first error record owned by a node. Otherwise, direct accesses to ERR<n>PFGCDN are RES0.

[ERR<n>FR](#) describes the features implemented by the node.

Attributes

ERR<n>PFGCDN is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | CDN | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

CDN, bits [31:0]

Countdown value.

This field is copied to Error Generation Counter when either:

- Software writes [ERR<n>PFGCTL](#).CDNEN with 1.
- The Error Generation Counter decrements to zero and [ERR<n>PFGCTL](#).R == 1.

While [ERR<n>PFGCTL](#).CDNEN == 1 and the Error Generation Counter is nonzero, the counter decrements by 1 for each cycle at an IMPLEMENTATION DEFINED clock rate. When the counter reaches 0, one of the errors enabled in the [ERR<n>PFGCTL](#) register is generated.

Note

The current Error Generation Counter value is not visible to software.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing ERR<n>PFGCDN

ERR<n>PFGCDN can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------------------|--------------|
| RAS | $0x810 + (64 * n)$ | ERR<n>PFGCDN |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERR<n>PFGCTL, Pseudo-fault Generation Control Register, n = 0 - 65534

The ERR<n>PFGCTL characteristics are:

Purpose

Enables controlled fault generation.

Configuration

This register is present only when error record <n> is implemented, the node implements the Common Fault Injection Model Extension (ERR<n>FR.INJ != 0b00) and error record <n> is the first error record owned by a node. Otherwise, direct accesses to ERR<n>PFGCTL are RES0.

[ERR<n>FR](#) describes the features implemented by the node.

Attributes

ERR<n>PFGCTL is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|------|----|----|----|----|----|----|----|----|----|----|--------|--------|------|----|----|------|------|-----|------|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| CDNEN | R | RES0 | | | | | | | | | | | RAO/WI | RAO/WI | PNER | CI | CE | DEUE | OUER | UEU | UCOF | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

CDNEN, bit [31]

Countdown Enable. Controls transfers of the value held in [ERR<n>PFGCDN](#) to the Error Generation Counter and enables this counter.

| CDNEN | Meaning |
|-------|---|
| 0b0 | The Error Generation Counter is disabled. |
| 0b1 | The Error Generation Counter is enabled. On a write of 1 to this field, the Error Generation Counter is set to ERR<n>PFGCDN.CDN . |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

R, bit [30]

When the node supports this control:

Restart. Controls whether the Error Generation Counter restarts or stops counting on reaching zero.

| R | Meaning |
|----------|---|
| 0b0 | On reaching zero, the Error Generation Counter will stop counting. |
| 0b1 | On reaching zero, the Error Generation Counter is set to ERR<n>PFGCDN.CDN . |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [29:13]

Reserved, RES0.

Bit [12]

When the node always sets [ERR<n>STATUS.MV](#) to 0b1 when an injected error is recorded:

Reserved, RAO/WI.

When the node supports this control:

Miscellaneous syndrome. The value written to [ERR<n>STATUS.MV](#) when an injected error is recorded.

| MV | Meaning |
|-----------|---|
| 0b0 | ERR<n>STATUS.MV is set to 0 when an injected error is recorded. |
| 0b1 | ERR<n>STATUS.MV is set to 1 when an injected error is recorded. |

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [11]

When the node always sets [ERR<n>STATUS.AV](#) to 0b1 when an injected error is recorded:

Reserved, RAO/WI.

When the node supports this control:

Address syndrome. The value written to [ERR<n>STATUS.AV](#) when an injected error is recorded.

| AV | Meaning |
|-----------|---|
| 0b0 | ERR<n>STATUS.AV is set to 0 when an injected error is recorded. |
| 0b1 | ERR<n>STATUS.AV is set to 1 when an injected error is recorded. |

The reset behavior of this field is:

- On an Error recovery reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

PN, bit [10]**When the node supports this control:**

Poison flag. The value written to [ERR<n>STATUS.PN](#) when an injected error is recorded.

| PN | Meaning |
|-----|---|
| 0b0 | ERR<n>STATUS.PN is set to 0 when an injected error is recorded. |
| 0b1 | ERR<n>STATUS.PN is set to 1 when an injected error is recorded. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

ER, bit [9]**When the node supports this control:**

Error Reported flag. The value written to [ERR<n>STATUS.ER](#) when an injected error is recorded.

| ER | Meaning |
|-----|---|
| 0b0 | ERR<n>STATUS.ER is set to 0 when an injected error is recorded. |
| 0b1 | ERR<n>STATUS.ER is set to 1 when an injected error is recorded. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CI, bit [8]**When the node supports this control:**

Critical Error flag. The value written to [ERR<n>STATUS.CI](#) when an injected error is recorded.

| CI | Meaning |
|-----|---|
| 0b0 | ERR<n>STATUS.CI is set to 0 when an injected error is recorded. |
| 0b1 | ERR<n>STATUS.CI is set to 1 when an injected error is recorded. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CE, bits [7:6]**When the node supports this control:**

Corrected Error generation enable. Controls the type of injected Corrected error generated by the fault injection feature of the node.

| CE | Meaning |
|------|--|
| 0b00 | An injected Corrected error will not be generated by the fault injection feature of the node. |
| 0b01 | An injected non-specific Corrected error is generated in the fault injection state. ERR<n>STATUS .CE is set to 0b10 when the injected error is recorded. |
| 0b10 | An injected transient Corrected error is generated in the fault injection state. ERR<n>STATUS .CE is set to 0b01 when the injected error is recorded. |
| 0b11 | An injected persistent Corrected error is generated in the fault injection state. ERR<n>STATUS .CE is set to 0b11 when the injected error is recorded. |

The set of permitted values for this field is defined by [ERR<n>PFGF](#).CE.

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

DE, bit [5]**When the node supports this control:**

Deferred Error generation enable. Controls whether an injected Deferred error is generated by the fault injection feature of the node.

| DE | Meaning |
|-----|---|
| 0b0 | An injected Deferred error will not be generated by the fault generation feature of the node. |
| 0b1 | An injected Deferred error is generated in the fault injection state. |

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UEO, bit [4]**When the node supports this control:**

Latent or Restartable Error generation enable. Controls whether an injected Latent or Restartable error is generated by the fault injection feature of the node.

| UEO | Meaning |
|-----|--|
| 0b0 | An injected Latent or Restartable error will not be generated by the fault generation feature of the node. |
| 0b1 | An injected Latent or Restartable error is generated in the fault injection state. |

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UER, bit [3]

When the node supports this control:

Signaled or Recoverable Error generation enable. Controls whether an injected Signaled or Recoverable error is generated by the fault injection feature of the node.

| UER | Meaning |
|-----|--|
| 0b0 | An injected Signaled or Recoverable error will not be generated by the fault generation feature of the node. |
| 0b1 | An injected Signaled or Recoverable error is generated in the fault injection state. |

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UEU, bit [2]

When the node supports this control:

Unrecoverable Error generation enable. Controls whether an injected Unrecoverable error is generated by the fault injection feature of the node.

| UEU | Meaning |
|-----|--|
| 0b0 | An injected Unrecoverable error will not be generated by the fault generation feature of the node. |
| 0b1 | An injected Unrecoverable error is generated in the fault injection state. |

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

UC, bit [1]**When the node supports this control:**

Uncontainable Error generation enable. Controls whether an injected Uncontainable error is generated by the fault injection feature of the node.

| UC | Meaning |
|-----|--|
| 0b0 | An injected Uncontainable error will not be generated by the fault generation feature of the node. |
| 0b1 | An injected Uncontainable error is generated in the fault injection state. |

The node enters the fault injection state when the Error Generation Counter decrements to zero. It is IMPLEMENTATION DEFINED whether the injected error is generated when the error is generated on an access to the component in the fault injection state and the data is not consumed.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

OF, bit [0]**When the node supports this control:**

Overflow flag. The value written to [ERR<n>STATUS](#).OF when an injected error is recorded.

| OF | Meaning |
|-----|--|
| 0b0 | ERR<n>STATUS .OF is set to 0 when an injected error is recorded. |
| 0b1 | ERR<n>STATUS .OF is set to 1 when an injected error is recorded. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing ERR<n>PFGCTL

ERR<n>PFGCTL can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|------------------|--------------|
| RAS | 0x808 + (64 * n) | ERR<n>PFGCTL |

Accesses on this interface are **RW**.

ERR<n>PFGF, Pseudo-fault Generation Feature Register, n = 0 - 65534

The ERR<n>PFGF characteristics are:

Purpose

Defines which common architecturally-defined fault generation features are implemented.

Configuration

This register is present only when error record <n> is implemented, the node implements the Common Fault Injection Model Extension (ERR<n>FR.INJ != 0b00) and error record <n> is the first error record owned by a node. Otherwise, direct accesses to ERR<n>PFGF are RES0.

[ERR<n>FR](#) describes the features implemented by the node.

Attributes

ERR<n>PFGF is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|-----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | R | SYN | NA | RES0 | | | | | | | | | | | | | | MV | AV | PN | ER | CI | CE | DE | UE | O | UE | R | UE | U | C | O | F |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:31]

Reserved, RES0.

R, bit [30]

Restartable. Support for Error Generation Counter restart mode.

| R | Meaning |
|-----|--|
| 0b0 | The node does not support this feature. ERR<n>PFGCTL.R is RES0. |
| 0b1 | Error Generation Counter restart mode is implemented and is controlled by ERR<n>PFGCTL.R . ERR<n>PFGCTL.R is a read/write field. |

SYN, bit [29]

Syndrome. Fault syndrome injection.

| SYN | Meaning |
|-----|---|
| 0b0 | When an injected error is recorded, the node sets ERR<n>STATUS .{IERR, SERR} to IMPLEMENTATION DEFINED values. ERR<n>STATUS .{IERR, SERR} are UNKNOWN when ERR<n>STATUS.V is 0. |
| 0b1 | When an injected error is recorded, the node does not update the ERR<n>STATUS .{IERR, SERR} fields. ERR<n>STATUS .{IERR, SERR} are writable when ERR<n>STATUS.V is 0. |

Note

If ERR<n>PFGF.SYN is 1, software can write specific values into the [ERR<n>STATUS](#).{IERR, SERR} fields when setting up a fault injection event. The sets of values that can be written to these fields is IMPLEMENTATION DEFINED.

NA, bit [28]

No access required. Defines whether this component fakes detection of the error on an access to the component or spontaneously in the fault injection state.

| NA | Meaning |
|-----|--|
| 0b0 | The component fakes detection of the error on an access to the component. |
| 0b1 | The component fakes detection of the error spontaneously in the fault injection state. |

Bits [27:13]

Reserved, RES0.

MV, bit [12]

Miscellaneous syndrome.

Additional syndrome injection. Defines whether software can control all or part of the syndrome recorded in the ERR<n>MISC<m> registers when an injected error is recorded.

| MV | Meaning |
|-----|--|
| 0b0 | When an injected error is recorded, the node might update ERR<n>MISC<m>. If any syndrome is recorded by the node in ERR<n>MISC<m>, then ERR<n>STATUS .MV is set to 1. ERR<n>PFGCTL .MV is RES0. |
| 0b1 | When an injected error is recorded, the node might update some, but not all ERR<n>MISC<m> syndrome fields, and updates ERR<n>STATUS .MV as follows: <ul style="list-style-type: none"> If any syndrome is recorded by the node in ERR<n>MISC<m>, then ERR<n>STATUS.MV is set to 1. Otherwise, ERR<n>STATUS.MV is set to ERR<n>PFGCTL.MV. <p>It is IMPLEMENTATION DEFINED which ERR<n>MISC<m> syndrome fields, if any, are updated. Some syndrome fields might always be updated by the node when an error is recorded. For example, a corrected error counter might always be updated when any countable error, including a countable injected error, is recorded. Other ERR<n>MISC<m> syndrome fields are not updated by the node and are writable when ERR<n>STATUS.MV is 0.</p> <p>If the node always sets ERR<n>STATUS.MV to 1 when recording an injected error then ERR<n>PFGCTL.MV is RAO/WI, otherwise ERR<n>PFGCTL.MV is a read/write field.</p> |

Note

If ERR<n>PFGF.MV is 1, software can write specific additional syndrome values into the ERR<n>MISC<m> registers when setting up a fault injection event. The values that can be written to these registers are IMPLEMENTATION DEFINED.

AV, bit [11]

Address syndrome. Address syndrome injection.

| AV | Meaning |
|-----|--|
| 0b0 | When an injected error is recorded, the node either sets ERR<n>ADDR and ERR<n>STATUS .AV for the access, or leaves these unchanged. ERR<n>PFGCTL .AV is RES0. |
| 0b1 | When an injected error is recorded, the node does not update ERR<n>ADDR and does one of: <ul style="list-style-type: none"> Sets ERR<n>STATUS.AV to ERR<n>PFGCTL.AV. ERR<n>PFGCTL.AV is a read/write field. Sets ERR<n>STATUS.AV to 1. ERR<n>PFGCTL.AV is RAO/WI. ERR<n>ADDR is writable when ERR<n>STATUS .AV is 0. |

Note

If ERR<n>PFGF.AV is 1, software can write a specific address value into [ERR<n>ADDR](#) when setting up a fault injection event.

PN, bit [10]

When the node supports this flag:

Poison flag. Describes how the fault generation feature of the node sets the [ERR<n>STATUS](#).PN status flag.

| PN | Meaning |
|-----|--|
| 0b0 | When an injected error is recorded, it is IMPLEMENTATION DEFINED whether the node sets ERR<n>STATUS .PN to 1. ERR<n>PFGCTL .PN is RES0. |
| 0b1 | When an injected error is recorded, ERR<n>STATUS .PN is set to ERR<n>PFGCTL .PN. ERR<n>PFGCTL .PN is a read/write field. |

This behavior replaces the architecture-defined rules for setting the [ERR<n>STATUS](#).PN bit.

Otherwise:

Reserved, RAZ.

ER, bit [9]

When the node supports this flag:

Error Reported flag. Describes how the fault generation feature of the node sets the [ERR<n>STATUS](#).ER status flag.

| ER | Meaning |
|-----|--|
| 0b0 | When an injected error is recorded, the node sets ERR<n>STATUS .ER according to the architecture-defined rules for setting the ER field. ERR<n>PFGCTL .ER is RES0. |
| 0b1 | When an injected error is recorded, ERR<n>STATUS .ER is set to ERR<n>PFGCTL .ER. This behavior replaces the architecture-defined rules for setting the ER bit. ERR<n>PFGCTL .ER is a read/write field. |

Otherwise:

Reserved, RAZ.

CI, bit [8]

When the node supports this flag:

Critical Error flag. Describes how the fault generation feature of the node sets the [ERR<n>STATUS](#).CI status flag.

| CI | Meaning |
|-----|--|
| 0b0 | When an injected error is recorded, it is IMPLEMENTATION DEFINED whether the node sets ERR<n>STATUS.CI to 1. ERR<n>PFGCTL.CI is RES0. |
| 0b1 | When an injected error is recorded, ERR<n>STATUS.CI is set to ERR<n>PFGCTL.CI . ERR<n>PFGCTL.CI is a read/write field. |

This behavior replaces the architecture-defined rules for setting the [ERR<n>STATUS.CI](#) bit.

Otherwise:

Reserved, RAZ.

CE, bits [7:6]

When the node supports this type of error:

Corrected Error generation. Describes the types of Corrected error that the fault generation feature of the node can generate.

| CE | Meaning |
|------|--|
| 0b00 | The fault generation feature of the node does not generate Corrected errors. ERR<n>PFGCTL.CE is RES0. |
| 0b01 | The fault generation feature of the node allows generation of a non-specific Corrected error, that is, a Corrected error that is recorded by setting ERR<n>STATUS.CE to 0b10. ERR<n>PFGCTL.CE is a read/write field. The values 0b10 and 0b11 in ERR<n>PFGCTL.CE are reserved. |
| 0b11 | The fault generation feature of the node allows generation of transient or persistent Corrected errors, that is, Corrected errors that are recorded by setting ERR<n>STATUS.CE to 0b01 or 0b11 respectively. ERR<n>PFGCTL.CE is a read/write field. The value 0b01 in ERR<n>PFGCTL.CE is reserved. |

All other values are reserved.

If [ERR<n>FR.FRX](#) is 1, then [ERR<n>FR.CE](#) indicates whether the node supports this type of error.

Otherwise:

Reserved, RAZ.

DE, bit [5]

When the node supports this type of error:

Deferred Error generation. Describes whether the fault generation feature of the node can generate Deferred errors.

| DE | Meaning |
|-----|---|
| 0b0 | The fault generation feature of the node does not generate Deferred errors. ERR<n>PFGCTL.DE is RES0. |
| 0b1 | The fault generation feature of the node allows generation of Deferred errors. ERR<n>PFGCTL.DE is a read/write field. |

If [ERR<n>FR.FRX](#) is 1, then [ERR<n>FR.DE](#) indicates whether the node supports this type of error.

Otherwise:

Reserved, RAZ.

UEO, bit [4]**When the node supports this type of error:**

Latent or Restartable Error generation. Describes whether the fault generation feature of the node can generate Latent or Restartable errors.

| UEO | Meaning |
|-----|--|
| 0b0 | The fault generation feature of the node does not generate Latent or Restartable errors. ERR<n>PFGCTL .UEO is RES0. |
| 0b1 | The fault generation feature of the node allows generation of Latent or Restartable errors. ERR<n>PFGCTL .UEO is a read/write field. |

If [ERR<n>FR](#).FRX is 1, then [ERR<n>FR](#).UEO indicates whether the node supports this type of error.

Otherwise:

Reserved, RAZ.

UER, bit [3]**When the node supports this type of error:**

Signaled or Recoverable Error generation. Describes whether the fault generation feature of the node can generate Signaled or Recoverable errors.

| UER | Meaning |
|-----|--|
| 0b0 | The fault generation feature of the node does not generate Signaled or Recoverable errors. ERR<n>PFGCTL .UER is RES0. |
| 0b1 | The fault generation feature of the node allows generation of Signaled or Recoverable errors. ERR<n>PFGCTL .UER is a read/write field. |

If [ERR<n>FR](#).FRX is 1, then [ERR<n>FR](#).UER indicates whether the node supports this type of error.

Otherwise:

Reserved, RAZ.

UEU, bit [2]**When the node supports this type of error:**

Unrecoverable Error generation. Describes whether the fault generation feature of the node can generate Unrecoverable errors.

| UEU | Meaning |
|-----|--|
| 0b0 | The fault generation feature of the node does not generate Unrecoverable errors. ERR<n>PFGCTL .UEU is RES0. |
| 0b1 | The fault generation feature of the node allows generation of Unrecoverable errors. ERR<n>PFGCTL .UEU is a read/write field. |

If [ERR<n>FR](#).FRX is 1, then [ERR<n>FR](#).UEU indicates whether the node supports this type of error.

Otherwise:

Reserved, RAZ.

UC, bit [1]**When the node supports this type of error:**

Uncontainable Error generation. Describes whether the fault generation feature of the node can generate Uncontainable errors.

| UC | Meaning |
|-----|---|
| 0b0 | The fault generation feature of the node does not generate Uncontainable errors. ERR<n>PFGCTL .UC is RES0. |
| 0b1 | The fault generation feature of the node allows generation of Uncontainable errors. ERR<n>PFGCTL .UC is a read/write field. |

If [ERR<n>FR](#).FRX is 1, then [ERR<n>FR](#).UC indicates whether the node supports this type of error.

Otherwise:

Reserved, RAZ.

OF, bit [0]**When the node supports this flag:**

Overflow flag. Describes how the fault generation feature of the node sets the [ERR<n>STATUS](#).OF status flag.

| OF | Meaning |
|-----|--|
| 0b0 | When an injected error is recorded, the node sets ERR<n>STATUS .OF according to the architecture-defined rules for setting the OF field. ERR<n>PFGCTL .OF is RES0. |
| 0b1 | When an injected error is recorded, ERR<n>STATUS .OF is set to ERR<n>PFGCTL .OF. This behavior replaces the architecture-defined rules for setting the OF bit. ERR<n>PFGCTL .OF is a read/write field. |

Otherwise:

Reserved, RAZ.

Accessing ERR<n>PFGF**ERR<n>PFGF can be accessed through the memory-mapped interfaces:**

| Component | Offset | Instance |
|-----------|------------------|------------|
| RAS | 0x800 + (64 * n) | ERR<n>PFGF |

Accesses on this interface are **RO**.

ERR<n>STATUS, Error Record Primary Status Register, n = 0 - 65534

The ERR<n>STATUS characteristics are:

Purpose

Contains status information for error record <n>, including:

- Whether any error has been detected (valid).
- Whether any detected error was not corrected, and returned to a Requester.
- Whether any detected error was not corrected and deferred.
- Whether an error record has been discarded because additional errors have been detected before the first error was handled by software (overflow).
- Whether any error has been reported.
- Whether the other error record registers contain valid information.
- Whether the error was reported because poison data was detected or because a corrupt value was detected by an error detection code.
- A primary error code.
- An IMPLEMENTATION DEFINED extended error code.

Within this register:

- ERR<n>STATUS.{AV, V, MV} are valid bits that define whether error record <n> registers are valid.
- ERR<n>STATUS.{UE, OF, CE, DE, UET} encode the types of error or errors recorded.
- ERR<n>STATUS.{CI, ER, PN, IERR, SERR} are syndrome fields.

Configuration

This register is present only when error record <n> is implemented. Otherwise, direct accesses to ERR<n>STATUS are RES0.

[ERR<q>FR](#) describes the features implemented by the node that owns error record <n>. <q> is the index of the first error record owned by the same node as error record <n>. If the node owns a single record, then q = n.

For IMPLEMENTATION DEFINED fields in ERR<n>STATUS, writing zero returns the error record to an initial quiescent state.

In particular, if any IMPLEMENTATION DEFINED syndrome fields might generate a Fault Handling or Error Recovery Interrupt request, writing zero is sufficient to deactivate the Interrupt request.

Fields that are read-only, non-zero, and ignore writes are compliant with this requirement.

Note

Arm recommends that any IMPLEMENTATION DEFINED syndrome field that can generate a Fault Handling, Error Recovery, Critical, or IMPLEMENTATION DEFINED, interrupt request is disabled at Cold reset and is enabled by software writing an IMPLEMENTATION DEFINED nonzero value to an IMPLEMENTATION DEFINED field in [ERR<q>CTLR](#).

Attributes

ERR<n>STATUS is a 64-bit register.

Field descriptions

When RAS System Architecture v1.1 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|-----|----|------|----|----|----|----|------|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| AV | V | UE | ER | OF | MV | CE | DE | PN | UET | CI | RES0 | | | | | IERR | | | | | | | SERR | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

AV, bit [31]

When error record <n> includes an address associated with an error:

Address Valid.

| AV | Meaning |
|-----|--|
| 0b0 | ERR<n>ADDR not valid. |
| 0b1 | ERR<n>ADDR contains an address associated with the highest priority error recorded by this record. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

V, bit [30]

Status Register Valid.

| V | Meaning |
|-----|---|
| 0b0 | ERR<n>STATUS not valid. |
| 0b1 | ERR<n>STATUS valid. At least one error has been recorded. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Access to this field is **W1C**.

UE, bit [29]

Uncorrected Error.

| UE | Meaning |
|-----|--|
| 0b0 | No errors have been detected, or all detected errors have been either corrected or deferred. |
| 0b1 | At least one detected error was not corrected and not deferred. |

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **W1C**.

ER, bit [28]

Error Reported.

| ER | Meaning |
|-----|---|
| 0b0 | No in-band error response (External Abort) signaled to the Requester making the access or other transaction. |
| 0b1 | An in-band error response was signaled by the component to the Requester making the access or other transaction. This can be because any of the following are true: <ul style="list-style-type: none"> • The applicable one of the ERR<q>CTLR.{WUE, RUE, UE} fields is implemented and was 1 when an error was detected and not corrected. • The applicable one of the ERR<q>CTLR.{WUE, RUE, UE} fields is not implemented and the component always reports errors. |

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as a Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester, causing this field to be set to 1. If no in-band error response to the Requester, this field is set to 0.

Note

An in-band error response signaled by the component might be masked and not generate any exception.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if all of the following are true:
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.DE == 0
 - this field can be set to 0b1 by a Deferred error
- **UNKNOWN/WI** if all of the following are true:
 - ERR<n>STATUS.UE == 0
 - this field is never set to 0b1 by a Deferred error
- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **W1C**.

OF, bit [27]

Overflow.

Indicates that multiple errors have been detected. This field is set to 1 when one of the following occurs:

- A Corrected error counter is implemented, an error is counted, and the counter overflows.
- ERR<n>STATUS.V was previously 1, a Corrected error counter is not implemented, and a Corrected error is recorded.
- ERR<n>STATUS.V was previously 1, and a type of error other than a Corrected error is recorded.

Otherwise, this field is unchanged when an error is recorded.

If a Corrected error counter is implemented:

- A direct write that modifies the counter overflow flag indirectly might set this field to an UNKNOWN value.

- A direct write to this field that clears this field to zero might indirectly set the counter overflow flag to an UNKNOWN value.

| OF | Meaning |
|-----|--|
| 0b0 | Since this field was last cleared to zero, no error syndrome has been discarded and, if a Corrected error counter is implemented, it has not overflowed. |
| 0b1 | Since this field was last cleared to zero, at least one error syndrome has been discarded or, if a Corrected error counter is implemented, it might have overflowed. |

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **W1C**.

MV, bit [26]

When error record <n> includes an additional information for an error:

Miscellaneous Registers Valid.

| MV | Meaning |
|-----|--|
| 0b0 | ERR<n>MISC<m> not valid. |
| 0b1 | The IMPLEMENTATION DEFINED contents of the ERR<n>MISC<m> registers contains additional information for an error recorded by this record. |

Note

If the ERR<n>MISC<m> registers can contain additional information for a previously recorded error, then the contents must be self-describing to software or a user. For example, certain fields might relate only to Corrected errors, and other fields only to the most recent error that was not discarded.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Access to this field is **W1C**.

Otherwise:

Reserved, RES0.

CE, bits [25:24]

Corrected Error.

| CE | Meaning |
|------|--|
| 0b00 | No errors were corrected. |
| 0b01 | At least one transient error was corrected. |
| 0b10 | At least one error was corrected. |
| 0b11 | At least one persistent error was corrected. |

The mechanism by which a component or node detects whether a Corrected error is transient or persistent is IMPLEMENTATION DEFINED. If no such mechanism is implemented, then the node sets this field to 0b10 when a corrected error is recorded.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **W1C**.

DE, bit [23]

Deferred Error.

| DE | Meaning |
|-----|--|
| 0b0 | No errors were deferred. |
| 0b1 | At least one error was not corrected and deferred. |

Support for deferring errors is IMPLEMENTATION DEFINED.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **W1C**.

PN, bit [22]

Poison.

| PN | Meaning |
|-----|--|
| 0b0 | Uncorrected error or Deferred error recorded because a corrupt value was detected, for example, by an error detection code (EDC), or Corrected error recorded. |
| 0b1 | Uncorrected error or Deferred error recorded because a poison value was detected. |

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- UNKNOWN/WI** if any of the following are true:
 - ERR<n>STATUS.V == 0
 - (ERR<n>STATUS.DE == 0 and ERR<n>STATUS.UE == 0)
- Otherwise, access to this field is **W1C**.

UET, bits [21:20]

Uncorrected Error Type. Describes the state of the component after detecting or consuming an Uncorrected error.

| UET | Meaning |
|------|---|
| 0b00 | Uncorrected error, Uncontainable error (UC). |
| 0b01 | Uncorrected error, Unrecoverable error (UEU). |
| 0b10 | Uncorrected error, Latent or Restartable error (UEO). |
| 0b11 | Uncorrected error, Signaled or Recoverable error (UER). |

Note

Software might use the information in the error record registers to determine what recovery is necessary.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - ERR<n>STATUS.V == 0
 - ERR<n>STATUS.UE == 0
- Otherwise, access to this field is **W1C**.

CI, bit [19]

Critical Error. Indicates whether a critical error condition has been recorded.

| CI | Meaning |
|-----|------------------------------|
| 0b0 | No critical error condition. |
| 0b1 | Critical error condition. |

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **W1C**.

Bits [18:16]

Reserved, RES0.

IERR, bits [15:8]

IMPLEMENTATION DEFINED error code. Used with any primary error code ERR<n>STATUS.SERR value. Further IMPLEMENTATION DEFINED information can be placed in the ERR<n>MISC<m> registers.

The implemented set of valid values that this field can take is IMPLEMENTATION DEFINED. If any value not in this set is written to this register, then the value read back from this field is UNKNOWN.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if all of the following are true:
 - the Common Fault Injection Model Extension is not implemented by the node that owns this error record

- ERR<n>STATUS.V == 0
- **UNKNOWN/WI** if all of the following are true:
 - ERR<q>PFGF.SYN == 0
 - ERR<n>STATUS.V == 0
- Otherwise, access to this field is **RW**.

SERR, bits [7:0]

Architecturally-defined primary error code. The primary error code might be used by a fault handling agent to triage an error without requiring device-specific code. For example, to count and threshold corrected errors in software, or generate a short log entry.

| SERR | Meaning |
|-------------|--|
| 0x00 | No error. |
| 0x01 | IMPLEMENTATION DEFINED error. |
| 0x02 | Data value from (non-associative) internal memory. For example, ECC from on-chip SRAM or buffer. |
| 0x03 | IMPLEMENTATION DEFINED pin. For example, nSEI pin. |
| 0x04 | Assertion failure. For example, consistency failure. |
| 0x05 | Error detected on internal data path. For example, parity on ALU result. |
| 0x06 | Data value from associative memory. For example, ECC error on cache data. |
| 0x07 | Address/control value from associative memory. For example, ECC error on cache tag. |
| 0x08 | Data value from a TLB. For example, ECC error on TLB data. |
| 0x09 | Address/control value from a TLB. For example, ECC error on TLB tag. |
| 0x0A | Data value from producer. For example, parity error on write data bus. |
| 0x0B | Address/control value from producer. For example, parity error on address bus. |
| 0x0C | Data value from (non-associative) external memory. For example, ECC error in SDRAM. |
| 0x0D | Illegal address (software fault). For example, access to unpopulated memory. |
| 0x0E | Illegal access (software fault). For example, byte write to word register. |
| 0x0F | Illegal state (software fault). For example, device not ready. |
| 0x10 | Internal data register. For example, parity on a SIMD&FP register. For a PE, all general-purpose, stack pointer, SIMD&FP, and SVE registers are data registers. |
| 0x11 | Internal control register. For example, Parity on a System register. For a PE, all registers other than general-purpose, stack pointer, SIMD&FP, and SVE registers are control registers. |
| 0x12 | Error response from Completer of access. For example, error response from cache write-back. |
| 0x13 | External timeout. For example, timeout on interaction with another component. |
| 0x14 | Internal timeout. For example, timeout on interface within the component. |
| 0x15 | Deferred error from Completer not supported at Requester. For example, poisoned data received from the Completer of an access by a Requester that cannot defer the error further. |
| 0x16 | Deferred error from Requester not supported at Completer. For example, poisoned data received from the Requester of an access by a Completer that cannot defer the error further. |
| 0x17 | Deferred error from Completer passed through. For example, poisoned data received from the Completer of an access and returned to the Requester. |
| 0x18 | Deferred error from Requester passed through. For example, poisoned data received from the Requester of an access and deferred to the Completer. |
| 0x19 | Error recorded by PCIe error logs. Indicates that the component has recorded an error in a PCIe error log. This might be the PCIe device status register, AER, DVSEC, or other mechanisms defined by PCIe. |
| 0x1A | Other internal error. For example, parity error on internal state of the component that is not covered by another primary error code. |

All other values are reserved.

The implemented set of valid values that this field can take is IMPLEMENTATION DEFINED. If any value not in this set is written to this register, then the value read back from this field is UNKNOWN.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- UNKNOWN/WI** if all of the following are true:
 - the Common Fault Injection Model Extension is not implemented by the node that owns this error record
 - ERR<n>STATUS.V == 0
- UNKNOWN/WI** if all of the following are true:
 - ERR<q>PFGF.SYN == 0
 - ERR<n>STATUS.V == 0
- Otherwise, access to this field is **RW**.

When RAS System Architecture v1.0 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|-----|------|----|----|----|----|----|------|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| AV | V | UE | ER | OF | MV | CE | DE | PN | UET | RES0 | | | | | | IERR | | | | | | SERR | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

AV, bit [31]

When error record <n> includes an address associated with an error:

Address Valid.

| AV | Meaning |
|-----|--|
| 0b0 | ERR<n>ADDR not valid. |
| 0b1 | ERR<n>ADDR contains an address associated with the highest priority error recorded by this record. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- RO** if all of the following are true:
 - ERR<n>STATUS.DE == 0
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.CE != 0b00
 - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write
- RO** if all of the following are true:
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.DE != 0
 - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write
- RO** if all of the following are true:
 - ERR<n>STATUS.UE != 0
 - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write
- Otherwise, access to this field is **W1C**.

Otherwise:

Reserved, RES0.

V, bit [30]

Status Register Valid.

| V | Meaning |
|-----|---|
| 0b0 | ERR<n>STATUS not valid. |
| 0b1 | ERR<n>STATUS valid. At least one error has been recorded. |

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- **RO** if all of the following are true:
 - ERR<n>STATUS.CE != 0b00
 - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write
- **RO** if all of the following are true:
 - ERR<n>STATUS.DE != 0
 - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write
- **RO** if all of the following are true:
 - ERR<n>STATUS.UE != 0
 - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write
- Otherwise, access to this field is **W1C**.

UE, bit [29]

Uncorrected Error.

| UE | Meaning |
|-----|--|
| 0b0 | No errors have been detected, or all detected errors have been either corrected or deferred. |
| 0b1 | At least one detected error was not corrected and not deferred. |

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- **RO** if all of the following are true:
 - ERR<n>STATUS.OF == 1
 - ERR<n>STATUS.OF is not being cleared to 0b0 in the same write
- Otherwise, access to this field is **W1C**.

ER, bit [28]

Error Reported.

| ER | Meaning |
|-----|---|
| 0b0 | No in-band error response (External Abort) signaled to the Requester making the access or other transaction. |
| 0b1 | An in-band error response was signaled by the component to the Requester making the access or other transaction. This can be because any of the following are true: <ul style="list-style-type: none"> • The applicable one of the ERR<q>CTLR.{WUE, RUE, UE} fields is implemented and was 1 when an error was detected and not corrected. • The applicable one of the ERR<q>CTLR.{WUE, RUE, UE} fields is not implemented and the component always reports errors. |

It is IMPLEMENTATION DEFINED whether an uncorrected error that is deferred and recorded as a Deferred error, but is not deferred to the Requester, will signal an in-band error response to the Requester, causing this field to be set to 1. If no in-band error response to the Requester, this field is set to 0.

Note

An in-band error response signaled by the component might be masked and not generate any exception.

If this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero, when any of:

- Clearing ERR<n>STATUS.V to 0.
- Clearing ERR<n>STATUS.UE to 0, if this field is never set to 1 by a Deferred error.
- Clearing ERR<n>STATUS.{UE,DE} to {0,0}, if this field can be set to 1 by a Deferred error.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if all of the following are true:
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.DE == 0
 - this field can be set to 0b1 by a Deferred error
- **UNKNOWN/WI** if all of the following are true:
 - ERR<n>STATUS.UE == 0
 - this field is never set to 0b1 by a Deferred error
- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- **RO** if all of the following are true:
 - ERR<n>STATUS.DE == 0
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.CE != 0b00
 - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write
- **RO** if all of the following are true:
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.DE != 0
 - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write
- **RO** if all of the following are true:
 - ERR<n>STATUS.UE != 0
 - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write
- Otherwise, access to this field is **W1C**.

OF, bit [27]

Overflow.

Indicates that multiple errors have been detected. This field is set to 1 when one of the following occurs:

- An Uncorrected error is detected and ERR<n>STATUS.UE == 1.
- A Deferred error is detected, ERR<n>STATUS.UE == 0 and ERR<n>STATUS.DE == 1.
- A Corrected error is detected, no Corrected error counter is implemented, ERR<n>STATUS.UE == 0, ERR<n>STATUS.DE == 0, and ERR<n>STATUS.CE != 0b00. ERR<n>STATUS.CE might be updated for the new Corrected error.
- A Corrected error counter is implemented, ERR<n>STATUS.UE == 0, ERR<n>STATUS.DE == 0, and the counter overflows.

It is IMPLEMENTATION DEFINED whether this field is set to 1 when one of the following occurs:

- A Deferred error is detected and ERR<n>STATUS.UE == 1.
- A Corrected error is detected, no Corrected error counter is implemented, and ERR<n>STATUS.{UE, DE} != {0, 0}.
- A Corrected error counter is implemented, ERR<n>STATUS.{UE, DE} != {0, 0}, and the counter overflows.

It is IMPLEMENTATION DEFINED whether this field is cleared to 0 when one of the following occurs:

- An Uncorrected error is detected and ERR<n>STATUS.UE == 0.
- A Deferred error is detected, ERR<n>STATUS.UE == 0, and ERR<n>STATUS.DE == 0.
- A Corrected error is detected, ERR<n>STATUS.UE == 0, ERR<n>STATUS.DE == 0, and ERR<n>STATUS.CE == 0b00.

The IMPLEMENTATION DEFINED clearing of this field might also depend on the value of the other error status fields.

If a Corrected error counter is implemented:

- A direct write that modifies the counter overflow flag indirectly might set this field to an UNKNOWN value.
- A direct write to this field that clears this field to 0 might indirectly set the counter overflow flag to an UNKNOWN value.

| OF | Meaning |
|---|--|
| 0b0 | <p>If ERR<n>STATUS.UE == 1, then no error syndrome for an Uncorrected error has been discarded.</p> <p>If ERR<n>STATUS.UE == 0 and ERR<n>STATUS.DE == 1, then no error syndrome for a Deferred error has been discarded.</p> <p>If ERR<n>STATUS.UE == 0, ERR<n>STATUS.DE == 0, and a Corrected error counter is implemented, then the counter has not overflowed.</p> <p>If ERR<n>STATUS.UE == 0, ERR<n>STATUS.DE == 0, ERR<n>STATUS.CE != 0b00, and no Corrected error counter is implemented, then no error syndrome for a Corrected error has been discarded.</p> |
| <p>Note</p> <p>This field might have been set to 1 when an error syndrome was discarded and later cleared to 0 when a higher priority syndrome was recorded.</p> | |
| 0b1 | At least one error syndrome has been discarded or, if a Corrected error counter is implemented, it might have overflowed. |

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **W1C**.

MV, bit [26]

When error record <n> includes an additional information for an error:

Miscellaneous Registers Valid.

| MV | Meaning |
|-----|--|
| 0b0 | ERR<n>MISC<m> not valid. |
| 0b1 | The IMPLEMENTATION DEFINED contents of the ERR<n>MISC<m> registers contains additional information for an error recorded by this record. |

Note

If the ERR<n>MISC<m> registers can contain additional information for a previously recorded error, then the contents must be self-describing to software or a user. For example, certain fields might relate only to Corrected errors, and other fields only to the most recent error that was not discarded.

The reset behavior of this field is:

- On a Cold reset, this field resets to 0.

Accessing this field has the following behavior:

- **RO** if all of the following are true:
 - ERR<n>STATUS.DE == 0
 - ERR<n>STATUS.UE == 0

- ERR<n>STATUS.CE != 0b00
- ERR<n>STATUS.CE is not being cleared to 0b00 in the same write
- **RO** if all of the following are true:
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.DE != 0
 - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write
- **RO** if all of the following are true:
 - ERR<n>STATUS.UE != 0
 - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write
- Otherwise, access to this field is **W1C**.

Otherwise:

Reserved, RES0.

CE, bits [25:24]

Corrected Error.

| CE | Meaning |
|------|--|
| 0b00 | No errors were corrected. |
| 0b01 | At least one transient error was corrected. |
| 0b10 | At least one error was corrected. |
| 0b11 | At least one persistent error was corrected. |

The mechanism by which a component or node detects whether a Corrected error is transient or persistent is IMPLEMENTATION DEFINED. If no such mechanism is implemented, then the node sets this field to 0b10 when a corrected error is recorded.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- **RO** if all of the following are true:
 - ERR<n>STATUS.OF == 1
 - ERR<n>STATUS.OF is not being cleared to 0b0 in the same write
- Otherwise, access to this field is **W1C**.

DE, bit [23]

Deferred Error.

| DE | Meaning |
|-----|--|
| 0b0 | No errors were deferred. |
| 0b1 | At least one error was not corrected and deferred. |

Support for deferring errors is IMPLEMENTATION DEFINED.

When clearing ERR<n>STATUS.V to 0, if this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- When ERR<n>STATUS.V == 0, access to this field is **UNKNOWN/WI**.
- **RO** if all of the following are true:
 - ERR<n>STATUS.OF == 1
 - ERR<n>STATUS.OF is not being cleared to 0b0 in the same write

- Otherwise, access to this field is **W1C**.

PN, bit [22]

Poison.

| PN | Meaning |
|-----|--|
| 0b0 | Uncorrected error or Deferred error recorded because a corrupt value was detected, for example, by an error detection code (EDC), or Corrected error recorded. |
| 0b1 | Uncorrected error or Deferred error recorded because a poison value was detected. |

If this field is nonzero, then Arm recommends that software write 1 to this field to clear this field to zero, when any of:

- Clearing ERR<n>STATUS.V to 0.
- Clearing both ERR<n>STATUS.{DE, UE} to 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- UNKNOWN/WI** if any of the following are true:
 - ERR<n>STATUS.V == 0
 - (ERR<n>STATUS.DE == 0 and ERR<n>STATUS.UE == 0)
- RO** if all of the following are true:
 - ERR<n>STATUS.DE == 0
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.CE != 0b00
 - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write
- RO** if all of the following are true:
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.DE != 0
 - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write
- RO** if all of the following are true:
 - ERR<n>STATUS.UE != 0
 - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write
- Otherwise, access to this field is **W1C**.

UET, bits [21:20]

Uncorrected Error Type. Describes the state of the component after detecting or consuming an Uncorrected error.

| UET | Meaning |
|------|---|
| 0b00 | Uncorrected error, Uncontainable error (UC). |
| 0b01 | Uncorrected error, Unrecoverable error (UEU). |
| 0b10 | Uncorrected error, Latent or Restartable error (UEO). |
| 0b11 | Uncorrected error, Signaled or Recoverable error (UER). |

Note

Software might use the information in the error record registers to determine what recovery is necessary.

If this field is nonzero, then Arm recommends that software write ones to this field to clear this field to zero, when any of:

- Clearing ERR<n>STATUS.V to 0.
- Clearing ERR<n>STATUS.UE to 0.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if any of the following are true:
 - ERR<n>STATUS.V == 0
 - ERR<n>STATUS.UE == 0
- **RO** if all of the following are true:
 - ERR<n>STATUS.DE == 0
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.CE != 0b00
 - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write
- **RO** if all of the following are true:
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.DE != 0
 - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write
- **RO** if all of the following are true:
 - ERR<n>STATUS.UE != 0
 - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write
- Otherwise, access to this field is **W1C**.

Bits [19:16]

Reserved, RES0.

IERR, bits [15:8]

IMPLEMENTATION DEFINED error code. Used with any primary error code ERR<n>STATUS.SERR value. Further IMPLEMENTATION DEFINED information can be placed in the ERR<n>MISC<m> registers.

The implemented set of valid values that this field can take is IMPLEMENTATION DEFINED. If any value not in this set is written to this register, then the value read back from this field is UNKNOWN.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if all of the following are true:
 - the Common Fault Injection Model Extension is not implemented by the node that owns this error record
 - ERR<n>STATUS.V == 0
- **UNKNOWN/WI** if all of the following are true:
 - ERR<q>PFGF.SYN == 0
 - ERR<n>STATUS.V == 0
- **RO** if all of the following are true:
 - ERR<n>STATUS.DE == 0
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.CE != 0b00
 - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write
- **RO** if all of the following are true:
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.DE != 0
 - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write
- **RO** if all of the following are true:
 - ERR<n>STATUS.UE != 0
 - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write
- Otherwise, access to this field is **RW**.

SERR, bits [7:0]

Architecturally-defined primary error code. The primary error code might be used by a fault handling agent to triage an error without requiring device-specific code. For example, to count and threshold corrected errors in software, or generate a short log entry.

| SERR | Meaning |
|-------------|--|
| 0x00 | No error. |
| 0x01 | IMPLEMENTATION DEFINED error. |
| 0x02 | Data value from (non-associative) internal memory. For example, ECC from on-chip SRAM or buffer. |
| 0x03 | IMPLEMENTATION DEFINED pin. For example, nSEI pin. |
| 0x04 | Assertion failure. For example, consistency failure. |
| 0x05 | Error detected on internal data path. For example, parity on ALU result. |
| 0x06 | Data value from associative memory. For example, ECC error on cache data. |
| 0x07 | Address/control value from associative memory. For example, ECC error on cache tag. |
| 0x08 | Data value from a TLB. For example, ECC error on TLB data. |
| 0x09 | Address/control value from a TLB. For example, ECC error on TLB tag. |
| 0x0A | Data value from producer. For example, parity error on write data bus. |
| 0x0B | Address/control value from producer. For example, parity error on address bus. |
| 0x0C | Data value from (non-associative) external memory. For example, ECC error in SDRAM. |
| 0x0D | Illegal address (software fault). For example, access to unpopulated memory. |
| 0x0E | Illegal access (software fault). For example, byte write to word register. |
| 0x0F | Illegal state (software fault). For example, device not ready. |
| 0x10 | Internal data register. For example, parity on a SIMD&FP register. For a PE, all general-purpose, stack pointer, SIMD&FP, and SVE registers are data registers. |
| 0x11 | Internal control register. For example, Parity on a System register. For a PE, all registers other than general-purpose, stack pointer, SIMD&FP, and SVE registers are control registers. |
| 0x12 | Error response from Completer of access. For example, error response from cache write-back. |
| 0x13 | External timeout. For example, timeout on interaction with another component. |
| 0x14 | Internal timeout. For example, timeout on interface within the component. |
| 0x15 | Deferred error from Completer not supported at Requester. For example, poisoned data received from the Completer of an access by a Requester that cannot defer the error further. |
| 0x16 | Deferred error from Requester not supported at Completer. For example, poisoned data received from the Requester of an access by a Completer that cannot defer the error further. |
| 0x17 | Deferred error from Completer passed through. For example, poisoned data received from the Completer of an access and returned to the Requester. |
| 0x18 | Deferred error from Requester passed through. For example, poisoned data received from the Requester of an access and deferred to the Completer. |
| 0x19 | Error recorded by PCIe error logs. Indicates that the component has recorded an error in a PCIe error log. This might be the PCIe device status register, AER, DVSEC, or other mechanisms defined by PCIe. |
| 0x1A | Other internal error. For example, parity error on internal state of the component that is not covered by another primary error code. |

All other values are reserved.

The implemented set of valid values that this field can take is IMPLEMENTATION DEFINED. If any value not in this set is written to this register, then the value read back from this field is UNKNOWN.

Note

This means that one or more bits of this field might be implemented as fixed read-as-zero or read-as-one values.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- **UNKNOWN/WI** if all of the following are true:
 - the Common Fault Injection Model Extension is not implemented by the node that owns this error record
 - ERR<n>STATUS.V == 0
- **UNKNOWN/WI** if all of the following are true:
 - ERR<q>PFGF.SYN == 0
 - ERR<n>STATUS.V == 0
- **RO** if all of the following are true:
 - ERR<n>STATUS.DE == 0
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.CE != 0b00
 - ERR<n>STATUS.CE is not being cleared to 0b00 in the same write
- **RO** if all of the following are true:
 - ERR<n>STATUS.UE == 0
 - ERR<n>STATUS.DE != 0
 - ERR<n>STATUS.DE is not being cleared to 0b0 in the same write
- **RO** if all of the following are true:
 - ERR<n>STATUS.UE != 0
 - ERR<n>STATUS.UE is not being cleared to 0b0 in the same write
- Otherwise, access to this field is **RW**.

Accessing ERR<n>STATUS

ERR<n>STATUS.{AV, V, UE, ER, OF, MV, CE, DE, PN, UET, CI} are write-one-to-clear (W1C) fields, meaning writes of zero are ignored, and a write of one or all-ones to the field clears the field to zero. ERR<n>STATUS.{IERR, SERR} are read/write (RW) fields, although the set of implemented valid values is IMPLEMENTATION DEFINED. See also [ERR<n>PFGF.SYN](#).

After reading ERR<n>STATUS, software must clear the valid fields in the register to allow new errors to be recorded. However, between reading the register and clearing the valid fields, a new error might have overwritten the register. To prevent this error being lost by software, the register prevents updates to fields that might have been updated by a new error.

When RAS System Architecture v1.0 is implemented:

- Writes to ERR<n>STATUS.{UE, DE, CE} are ignored if ERR<n>STATUS.OF is 1 and is not being cleared to 0.
- Writes to ERR<n>STATUS.V are ignored if any of ERR<n>STATUS.{UE, DE, CE} are nonzero and are not being cleared to zero.
- Writes to ERR<n>STATUS.{AV, MV} and the ERR<n>STATUS.{ER, PN, UET, IERR, SERR} syndrome fields are ignored if the highest priority nonzero error status field is not being cleared to zero. The error status fields in priority order from highest to lowest, are ERR<n>STATUS.UE, ERR<n>STATUS.DE, and ERR<n>STATUS.CE.

When RAS System Architecture v1.1 is implemented, a write to the register is ignored if all of:

- Any of ERR<n>STATUS.{V, UE, OF, CE, DE} are nonzero before the write.
- The write does not clear the nonzero ERR<n>STATUS.{V, UE, OF, CE, DE} fields to zero by writing ones to the applicable field or fields.

Some of the fields in ERR<n>STATUS are also defined as UNKNOWN where certain combinations of ERR<n>STATUS.{V, DE, UE} are zero. The rules for writes to ERR<n>STATUS allow a node to implement such a field as a fixed read-only value.

For example, when RAS System Architecture v1.1 is implemented, a write to ERR<n>STATUS when ERR<n>STATUS.V is 1 results in either ERR<n>STATUS.V field being cleared to zero, or ERR<n>STATUS.V not changing. Since all fields in ERR<n>STATUS, other than ERR<n>STATUS.{AV, V, MV}, usually read as UNKNOWN values when ERR<n>STATUS.V is zero, this means those fields can be implemented as read-only if applicable.

To ensure correct and portable operation, when software is clearing the valid fields in the register to allow new errors to be recorded, Arm recommends that software:

- Read ERR<n>STATUS and determine which fields need to be cleared to zero.
- Write ones to all the W1C fields that are nonzero in the read value.

- Write zero to all the W1C fields that are zero in the read value.
- Write zero to all the RW fields.

Otherwise, these fields might not have the correct value when a new fault is recorded.

An exception is when the node supports writing to these fields as part of fault injection. See also [ERR<n>PFGF.SYN](#).

ERR<n>STATUS can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|---------------------------|--------------|
| RAS | $0 \times 010 + (64 * n)$ | ERR<n>STATUS |

This interface is accessible as follows:

- When ERR<n>STATUS.V != 0, ERR<n>STATUS.V is not being cleared to 0b0 in the same write and RAS System Architecture v1.1 is implemented accesses to this register are **RO**.
- When ERR<n>STATUS.UE != 0, ERR<n>STATUS.UE is not being cleared to 0b0 in the same write and RAS System Architecture v1.1 is implemented accesses to this register are **RO**.
- When ERR<n>STATUS.OF != 0, ERR<n>STATUS.OF is not being cleared to 0b0 in the same write and RAS System Architecture v1.1 is implemented accesses to this register are **RO**.
- When ERR<n>STATUS.CE != 0b00, ERR<n>STATUS.CE is not being cleared to 0b00 in the same write and RAS System Architecture v1.1 is implemented accesses to this register are **RO**.
- When ERR<n>STATUS.DE != 0, ERR<n>STATUS.DE is not being cleared to 0b0 in the same write and RAS System Architecture v1.1 is implemented accesses to this register are **RO**.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRPIDR0, Peripheral Identification Register 0

The ERRPIDR0 characteristics are:

Purpose

Provides discovery information about the component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

ERRPIDR0 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRPIDR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | PART_0 | | | | | | | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, bits [7:0].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, it is stored in [ERRPIDR1](#).PART_1 and ERRPIDR0.PART_0. There are 8 bits, [ERRPIDR2](#).REVISION and [ERRPIDR3](#).REVAND, available to define the revision of the component.
- If a 16-bit part number is used, it is stored in [ERRPIDR2](#).PART_2, [ERRPIDR1](#).PART_1 and ERRPIDR0.PART_0. There are 4 bits, [ERRPIDR3](#).REVISION, available to define the revision of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRPIDR0

ERRPIDR0 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| RAS | 0xFE0 | ERRPIDR0 |

Accesses on this interface are **RO**.

ERRPIDR1, Peripheral Identification Register 1

The ERRPIDR1 characteristics are:

Purpose

Provides discovery information about the component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

ERRPIDR1 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRPIDR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---|--------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | DES_0 | | | | PART_1 | | | |

Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, JEP106 identification code, bits [3:0]. ERRPIDR1.DES_0 and [ERRPIDR2.DES_1](#) together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be not the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC <http://www.jedec.org>.

Note

For a component designed by Arm Limited, the JEP106 identification code is 0x3B.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PART_1, bits [3:0]

Part number, bits [11:8].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, it is stored in ERRPIDR1.PART_1 and [ERRPIDR0.PART_0](#). There are 8 bits, [ERRPIDR2.REVISION](#) and [ERRPIDR3.REVAND](#), available to define the revision of the component.

- If a 16-bit part number is used, it is stored in [ERRPIDR2](#).PART_2, ERRPIDR1.PART_1 and [ERRPIDR0](#).PART_0. There are 4 bits, [ERRPIDR3](#).REVISION, available to define the revision of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRPIDR1

ERRPIDR1 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| RAS | 0xFE4 | ERRPIDR1 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRPIDR2, Peripheral Identification Register 2

The ERRPIDR2 characteristics are:

Purpose

Provides discovery information about the component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

ERRPIDR2 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRPIDR2 is a 32-bit register.

Field descriptions

When the component uses a 12-bit part number:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|-------|----|-------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | REVISION | | | | JEDEC | | DES 1 | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Component major revision. ERRPIDR2.REVISION and [ERRPIDR3.REVAND](#) together form the revision number of the component, with ERRPIDR2.REVISION being the most significant part and [ERRPIDR3.REVAND](#) the least significant part. When a component is changed, ERRPIDR2.REVISION or [ERRPIDR3.REVAND](#) are increased to ensure that software can differentiate the different revisions of the component. [ERRPIDR3.REVAND](#) should be set to 0b0000 when ERRPIDR2.REVISION is increased.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used.

Reads as 0b1.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, JEP106 identification code, bits [6:4]. [ERRPIDR1.DES_0](#) and ERRPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be the same as the

implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC <http://www.jedec.org>.

Note

For a component designed by Arm Limited, the JEP106 identification code is 0x3B.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

When the component uses a 16-bit part number:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|-------|----|-------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | PART_2 | | | | JEDEC | | DES_1 | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

PART_2, bits [7:4]

Part number, bits [15:12].

The part number is selected by the designer of the component. The designer chooses whether to use a 12-bit or a 16-bit part number:

- If a 12-bit part number is used, it is stored in [ERRPIDR1.PART_1](#) and [ERRPIDR0.PART_0](#). There are 8 bits, [ERRPIDR2.REVISION](#) and [ERRPIDR3.REVAND](#), available to define the revision of the component.
- If a 16-bit part number is used, it is stored in [ERRPIDR2.PART_2](#), [ERRPIDR1.PART_1](#) and [ERRPIDR0.PART_0](#). There are 4 bits, [ERRPIDR3.REVISION](#), available to define the revision of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used.

Reads as 0b1.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, JEP106 identification code, bits [6:4]. [ERRPIDR1.DES_0](#) and [ERRPIDR2.DES_1](#) together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC <http://www.jedec.org>.

Note

For a component designed by Arm Limited, the JEP106 identification code is 0x3B.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRPIDR2

ERRPIDR2 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| RAS | 0xFE8 | ERRPIDR2 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

ERRPIDR3, Peripheral Identification Register 3

The ERRPIDR3 characteristics are:

Purpose

Provides discovery information about the component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

ERRPIDR3 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRPIDR3 is a 32-bit register.

Field descriptions

When the component uses a 12-bit part number:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | REVAND | | | | CMOD | | | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Component minor revision. [ERRPIDR2.REVISION](#) and ERRPIDR3.REVAND together form the revision number of the component, with [ERRPIDR2.REVISION](#) being the most significant part and ERRPIDR3.REVAND the least significant part. When a component is changed, [ERRPIDR2.REVISION](#) or ERRPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. ERRPIDR3.REVAND should be set to 0b0000 when [ERRPIDR2.REVISION](#) is increased.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer Modified.

Indicates the component has been modified.

A value of 0b0000 means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

For any two components with the same Unique Component Identifier:

- If the value of the CMOD fields of both components equals zero, the components are identical.
- If the CMOD fields of both components have the same non-zero value, it does not necessarily mean that they have the same modifications.

- If the value of the CMOD field of either of the two components is non-zero, they might not be identical, even though they have the same Unique Component Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

When the component uses a 16-bit part number:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | REVISION | | | | CMOD | | | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Component revision. When a component is changed, ERRPIDR3.REVISION is increased to ensure that software can differentiate the different revisions of the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer Modified.

Indicates the component has been modified.

A value of 0b0000 means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

For any two components with the same Unique Component Identifier:

- If the value of the CMOD fields of both components equals zero, the components are identical.
- If the CMOD fields of both components have the same non-zero value, it does not necessarily mean that they have the same modifications.
- If the value of the CMOD field of either of the two components is non-zero, they might not be identical, even though they have the same Unique Component Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRPIDR3

ERRPIDR3 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| RAS | 0xFEC | ERRPIDR3 |

Accesses on this interface are **RO**.

ERRPIDR4, Peripheral Identification Register 4

The ERRPIDR4 characteristics are:

Purpose

Provides discovery information about the component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

ERRPIDR4 is implemented only as part of a memory-mapped group of error records.

Attributes

ERRPIDR4 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|------|---|---|---|-------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | SIZE | | | | DES_2 | | | |

Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

Size of the component.

The distance from the start of the address space used by this component to the end of the component identification registers.

A value of 0b0000 means one of the following is true:

- The component uses a single 4KB block.
- The component uses an IMPLEMENTATION DEFINED number of 4KB blocks.

Any other value means the component occupies $2^{\text{ERRPIDR4.SIZE}}$ 4KB blocks.

Using this field to indicate the size of the component is deprecated. This field might not correctly indicate the size of the component. Arm recommends that software determine the size of the component from the Unique Component Identifier fields, and other IMPLEMENTATION DEFINED registers in the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

DES_2, bits [3:0]

Designer, JEP106 continuation code. This is the JEDEC-assigned JEP106 bank identifier for the designer of the component, minus 1. The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC <http://www.jedec.org>.

Note

For a component designed by Arm Limited, the JEP106 bank is 5, meaning this field has the value 0x4.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing ERRPIDR4

ERRPIDR4 can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| RAS | 0xFD0 | ERRPIDR4 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_ABPR, CPU Interface Aliased Binary Point Register

The GICC_ABPR characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_ABPR are RES0.

In systems that support two Security states:

- This register is an alias of the Non-secure copy of [GICC_BPR](#).
- Non-secure accesses to this register return a shifted value of the binary point.
- If [ICC_CTLR_EL3](#).CBPR_EL1NS == 1, Secure accesses to this register access [ICC_BPR0_EL1](#).

Attributes

GICC_ABPR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|--------------|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Binary_Point | |

Bits [31:3]

Reserved, RES0.

Binary_Point, bits [2:0]

Controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. The following list describes how this field determines the interrupt priority bits assigned to the group priority field:

- 'Secure ICC_BPR1_EL1 Binary Point when CBPR == 0' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069), for the processing of Group 1 interrupts in a GIC implementation that supports interrupt grouping, when [GICC_CTLR](#).CBPR == 0.
- 'Non-secure ICC_BPR1_EL1 Binary Point when CBPR == 0' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069), for all other cases.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GICC_ABPR

This register is used only when System register access is not enabled. When System register access is enabled, the System registers [ICC_BPR0_EL1](#) and [ICC_BPR1_EL1](#) provide equivalent functionality.

GICC_ABPR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|--------|-----------|
| GIC CPU interface | 0x001C | GICC_ABPR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_AEOIR, CPU Interface Aliased End Of Interrupt Register

The GICC_AEOIR characteristics are:

Purpose

A write to this register performs priority drop for the specified Group 1 interrupt and, if the appropriate [GICC_CTLR.EOImodeS](#) or [GICC_CTLR.EOImodeNS](#) field == 0, also deactivates the interrupt.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_AEOIR are RES0.

When [GICD_CTLR.DS](#)==0, this register is an alias of the Non-secure view of [GICC_EOIR](#). A Secure access to this register is identical to a Non-secure access to [GICC_EOIR](#).

Attributes

GICC_AEOIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Accessing GICC_AEOIR

A write to this register must correspond to the most recently acknowledged Group 1 interrupt. If a value other than the last value read from [GICC_AIAR](#) is written to this register, the effect is UNPREDICTABLE.

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_EOIR1](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_EOIR1_EL1](#) provides equivalent functionality.

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_AEOIR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|--------|------------|
| GIC CPU interface | 0x0024 | GICC_AEOIR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **WO**.
- When an access is Secure accesses to this register are **WO**.
- When an access is Non-secure accesses to this register are **WO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_AHPPIR, CPU Interface Aliased Highest Priority Pending Interrupt Register

The GICC_AHPPIR characteristics are:

Purpose

If the highest priority pending interrupt is in Group 1, this register provides the INTID of the highest priority pending interrupt on the CPU interface.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_AHPPIR are RES0.

If [GICD_CTLR.DS](#)=0, this register is an alias of the Non-secure view of [GICC_HPPIR](#). A Secure access to this register is identical to a Non-secure access to [GICC_HPPIR](#).

Attributes

GICC_AHPPIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Accessing GICC_AHPPIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_HPPIR1](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_HPPIR1_EL1](#) provides equivalent functionality.

If the highest priority pending interrupt is in Group 0, a read of this register returns the special INTID 1023.

Interrupt identifiers corresponding to an interrupt group that is not enabled are ignored.

If the highest priority pending interrupt is a direct interrupt that is both individually enabled in the Distributor and part of an interrupt group that is enabled in the Distributor, and the interrupt group is disabled in the CPU interface for this PE, this register returns the special INTID 1023.

For more information about pending interrupts that are not considered when determining the highest priority pending interrupt, see 'Preemption' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_AHPPIR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|--------|-------------|
| GIC CPU interface | 0x0028 | GICC_AHPPIR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_AIAR, CPU Interface Aliased Interrupt Acknowledge Register

The GICC_AIAR characteristics are:

Purpose

Provides the INTID of the signaled Group 1 interrupt. A read of this register by the PE acts as an acknowledge for the interrupt.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_AIAR are RES0.

When [GICD_CTLR.DS](#)==0, this register is an alias of the Non-secure view of [GICC_IAR](#). A Secure access to this register is identical to a Non-secure access to [GICC_IAR](#).

Attributes

GICC_AIAR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Accessing GICC_AIAR

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_AIAR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
|-----------|--------|----------|

| | | |
|----------------------|--------|-----------|
| GIC CPU interface | 0x0020 | GICC_AIAR |
|----------------------|--------|-----------|

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_APR<n>, CPU Interface Active Priorities Registers, n = 0 - 3

The GICC_APR<n> characteristics are:

Purpose

Provides information about interrupt active priorities.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_APR<n> are RES0.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

When [GICD_CTLR.DS](#) == 0, these registers are Banked, and Non-secure accesses do not affect Secure operation. The Secure copies of these registers hold active priorities for Group 0 interrupts, and the Non-secure copies provide a Non-secure view of the active priorities for Group 1 interrupts.

GICC_APR1 is only implemented in implementations that support 6 or more bits of priority. GICC_APR2 and GICC_APR3 are only implemented in implementations that support 7 bits of priority.

When [GICD_CTLR.DS](#) == 1, these registers hold the active priorities for Group 0 interrupts, and the active priorities for Group 1 interrupts are held by the [GICC_NSAPR<n>](#) registers.

Attributes

GICC_APR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing GICC_APR<n>

These registers are used only when System register access is not enabled. When System register access is enabled the following registers provide equivalent functionality:

- In AArch64:
 - For Group 0, [ICC_AP0R<n>_EL1](#).
 - For Group 1, [ICC_AP1R<n>_EL1](#).
- In AArch32:
 - For Group 0, [ICC_AP0R<n>](#).
 - For Group 1, [ICC_AP1R<n>](#).

GICC_APR<n> can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|---------------------------|-------------|
| GIC CPU interface | $0 \times 00D0 + (4 * n)$ | GICC_APR<n> |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_BPR, CPU Interface Binary Point Register

The GICC_BPR characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_BPR are RES0.

In systems that support two Security states:

- This register is Banked.
- The Secure instance of this register determines Group 0 interrupt preemption.
- The Non-secure instance of this register determines Group 1 interrupt preemption.

In systems that support only one Security state, when [GICC_CTLR](#).CBPR == 0, this register determines only Group 0 interrupt preemption.

When [GICC_CTLR](#).CBPR == 1, this register determines interrupt preemption for both Group 0 and Group 1 interrupts.

Attributes

GICC_BPR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | Binary Point | | | | | | | | | | | | | |

Bits [31:3]

Reserved, RES0.

Binary_Point, bits [2:0]

Controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field. The following list describes how this field determines the interrupt priority bits assigned to the group priority field:

- 'Secure ICC_BPR1_EL1 Binary Point when CBPR == 0' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069), for the processing of Group 1 interrupts in a GIC implementation that supports interrupt grouping, when [GICC_CTLR](#).CBPR == 0.
- 'Non-secure ICC_BPR1_EL1 Binary Point when CBPR == 0' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069), for all other cases.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Note

Aliasing the Non-secure GICC_BPR as [GICC_ABPR](#) in a multiprocessor system permits a PE that can make only Secure accesses to configure the preemption setting for Group 1 interrupts by accessing [GICC_ABPR](#).

Accessing GICC_BPR

This register is used only when System register access is not enabled. When System register access is enabled this register is RAZ/WI, and the System registers [ICC_BPR0_EL1](#) and [ICC_BPR1_EL1](#) provide equivalent functionality.

GICC_BPR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|--------|----------|
| GIC CPU interface | 0x0008 | GICC_BPR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_CTLR, CPU Interface Control Register

The GICC_CTLR characteristics are:

Purpose

Controls the CPU interface, including enabling of interrupt groups, interrupt signal bypass, binary point registers used, and separation of priority drop and interrupt deactivation.

Note

If the GIC implementation supports two Security states, independent EOI controls are provided for accesses from each Security state. Secure accesses handle both Group 0 and Group 1 interrupts, and Non-secure accesses handle Group 1 interrupts only.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_CTLR are RES0.

In a GIC implementation that supports two Security states:

- This register is Banked.
- The register bit assignments are different in the Secure and Non-secure copies.

Attributes

GICC_CTLR is a 32-bit register.

Field descriptions

When GICD_CTLR.DS==0, Non-secure access:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|-----------|----|------|----|---------------|----|---------------|----|------|----|------------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | EOImodeNS | | RES0 | | IRQBypDisGrp1 | | FIQBypDisGrp1 | | RES0 | | EnableGrp1 | | | | | | | | | | | |

Bits [31:10]

Reserved, RES0.

EOImodeNS, bit [9]

Controls the behavior of Non-secure accesses to [GICC_EOIR](#), [GICC_AEOIR](#), and [GICC_DIR](#).

| EOImodeNS | Meaning |
|-----------|---|
| 0b0 | GICC_EOIR and GICC_AEOIR provide both priority drop and interrupt deactivation functionality. Accesses to GICC_DIR are UNPREDICTABLE. |
| 0b1 | GICC_EOIR and GICC_AEOIR provide priority drop functionality only. GICC_DIR provides interrupt deactivation functionality. |

Note

An implementation is permitted to make this bit RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [8:7]

Reserved, RES0.

IRQBypDisGrp1, bit [6]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 1:

| IRQBypDisGrp1 | Meaning |
|---------------|--|
| 0b0 | The bypass IRQ signal is signaled to the PE. |
| 0b1 | The bypass IRQ signal is not signaled to the PE. |

If System register access is enabled for EL3 and [ICC_SRE_EL3.DIB](#) == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

FIQBypDisGrp1, bit [5]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 1:

| FIQBypDisGrp1 | Meaning |
|---------------|--|
| 0b0 | The bypass FIQ signal is signaled to the PE. |
| 0b1 | The bypass FIQ signal is not signaled to the PE. |

If System register access is enabled for EL3 and [ICC_SRE_EL3.DFB](#) == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bits [4:1]

Reserved, RES0.

EnableGrp1, bit [0]

This Non-secure field enables the signaling of Group 1 interrupts by the CPU interface to a target PE:

| EnableGrp1 | Meaning |
|------------|--|
| 0b0 | Group 1 interrupt signaling is disabled. |
| 0b1 | Group 1 interrupt signaling is enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

When GICD_CTLR.DS==0, Secure access:

| | | | | | |
|--|-----------|----------|---------------|---------------|---------------|
| 313029282726252423222120191817161514131211 | 10 | 9 | 8 | 7 | 6 |
| RES0 | EOImodeNS | EOImodeS | IRQBypDisGrp1 | FIQBypDisGrp1 | IRQBypDisGrp0 |

Bits [31:11]

Reserved, RES0.

EOImodeNS, bit [10]

Controls the behavior of Non-secure accesses to [GICC_EOIR](#), [GICC_AEOIR](#), and [GICC_DIR](#).

| EOImodeNS | Meaning |
|-----------|---|
| 0b0 | GICC_EOIR and GICC_AEOIR provide both priority drop and interrupt deactivation functionality. Accesses to GICC_DIR are UNPREDICTABLE. |
| 0b1 | GICC_EOIR and GICC_AEOIR provide priority drop functionality only. GICC_DIR provides interrupt deactivation functionality. |

Note

An implementation is permitted to make this bit RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

EOImodeS, bit [9]

Controls the behavior of Secure accesses to [GICC_EOIR](#), [GICC_AEOIR](#), and [GICC_DIR](#).

| EOImodeS | Meaning |
|----------|---|
| 0b0 | GICC_EOIR and GICC_AEOIR provide both priority drop and interrupt deactivation functionality. Accesses to GICC_DIR are UNPREDICTABLE. |
| 0b1 | GICC_EOIR and GICC_AEOIR provide priority drop functionality only. GICC_DIR provides interrupt deactivation functionality. |

Note

An implementation is permitted to make this bit RAO/WI.

This field shares state with [GICC_CTLR](#).EOImode.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

IRQBypDisGrp1, bit [8]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 1:

| IRQBypDisGrp1 | Meaning |
|---------------|--|
| 0b0 | The bypass IRQ signal is signaled to the PE. |
| 0b1 | The bypass IRQ signal is not signaled to the PE. |

If System register access is enabled for EL3 and [ICC_SRE_EL3](#).DIB == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

FIQByDisGrp1, bit [7]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 1:

| FIQByDisGrp1 | Meaning |
|--------------|--|
| 0b0 | The bypass FIQ signal is signaled to the PE. |
| 0b1 | The bypass FIQ signal is not signaled to the PE. |

If System register access is enabled for EL3 and [ICC_SRE_EL3.DFB](#) == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

IRQByDisGrp0, bit [6]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 0:

| IRQByDisGrp0 | Meaning |
|--------------|--|
| 0b0 | The bypass IRQ signal is signaled to the PE. |
| 0b1 | The bypass IRQ signal is not signaled to the PE. |

If System register access is enabled for EL3 and [ICC_SRE_EL3.DIB](#) == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

FIQByDisGrp0, bit [5]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 0:

| FIQByDisGrp0 | Meaning |
|--------------|--|
| 0b0 | The bypass FIQ signal is signaled to the PE. |
| 0b1 | The bypass FIQ signal is not signaled to the PE. |

If System register access is enabled for EL3 and [ICC_SRE_EL3.DIB](#) == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

CBPR, bit [4]

Controls whether [GICC_BPR](#) provides common control of preemption to Group 0 and Group 1 interrupts:

| CBPR | Meaning |
|------|--|
| 0b0 | GICC_BPR determines preemption for Group 0 interrupts only. |
| 0b1 | GICC_ABPR determines preemption for Group 1 interrupts. GICC_BPR determines preemption for both Group 0 and Group 1 interrupts. |

This field is an alias of [ICC_CTLR_EL3.CBPR_EL1NS](#).

In a GIC that supports two Security states, when CBPR == 1:

- A Non-secure read of [GICC_BPR](#) returns the value of Secure [GICC_BPR](#).Binary_Point, incremented by 1, and saturated to 0b111.
- Non-secure writes of [GICC_BPR](#) are ignored.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

FIQEn, bit [3]

Controls whether the CPU interface signals Group 0 interrupts to a target PE using the FIQ or IRQ signal:

| FIQEn | Meaning |
|-------|---|
| 0b0 | Group 0 interrupts are signaled using the IRQ signal. |
| 0b1 | Group 0 interrupts are signaled using the FIQ signal. |

Group 1 interrupts are signaled using the IRQ signal only.

If an implementation supports two Security states, this bit is permitted to be RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bit [2]

Reserved, RES0.

EnableGrp1, bit [1]

This Non-secure field enables the signaling of Group 1 interrupts by the CPU interface to a target PE:

| EnableGrp1 | Meaning |
|------------|--|
| 0b0 | Group 1 interrupt signaling is disabled. |
| 0b1 | Group 1 interrupt signaling is enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

EnableGrp0, bit [0]

Enables the signaling of Group 0 interrupts by the CPU interface to a target PE:

| EnableGrp0 | Meaning |
|------------|--|
| 0b0 | Group 0 interrupt signaling is disabled. |
| 0b1 | Group 0 interrupt signaling is enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

When GICD_CTLR.DS == 1:

| | | | | | |
|--|---------|---------------|---------------|---------------|---------------|
| 31302928272625242322212019181716151413121110 | 9 | 8 | 7 | 6 | 5 |
| RES0 | EOImode | IRQBypDisGrp1 | FIQBypDisGrp1 | IRQBypDisGrp0 | FIQBypDisGrp0 |

Bits [31:10]

Reserved, RES0.

EOImode, bit [9]

Controls the behavior of accesses to [GICC_EOIR](#), [GICC_AEOIR](#), and [GICC_DIR](#).

| EOImode | Meaning |
|---------|---|
| 0b0 | GICC_EOIR and GICC_AEOIR provide both priority drop and interrupt deactivation functionality. Accesses to GICC_DIR are UNPREDICTABLE. |
| 0b1 | GICC_EOIR and GICC_AEOIR provide priority drop functionality only. GICC_DIR provides interrupt deactivation functionality. |

Note

An implementation is permitted to make this bit RAO/WI.

This field shares state with [GICC_CTLR.EOImodeS](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

IRQBypDisGrp1, bit [8]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 1:

| IRQBypDisGrp1 | Meaning |
|---------------|--|
| 0b0 | The bypass IRQ signal is signaled to the PE. |
| 0b1 | The bypass IRQ signal is not signaled to the PE. |

If System register access is enabled for EL3 and [ICC_SRE_EL3.DIB](#) == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

FIQBypDisGrp1, bit [7]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 1:

| FIQBypDisGrp1 | Meaning |
|----------------------|--|
| 0b0 | The bypass FIQ signal is signaled to the PE. |
| 0b1 | The bypass FIQ signal is not signaled to the PE. |

If System register access is enabled for EL3 and [ICC_SRE_EL3.DFB](#) == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

IRQBypDisGrp0, bit [6]

When the signaling of IRQs by the CPU interface is disabled, this field partly controls whether the bypass IRQ signal is signaled to the PE for Group 0:

| IRQBypDisGrp0 | Meaning |
|----------------------|--|
| 0b0 | The bypass IRQ signal is signaled to the PE. |
| 0b1 | The bypass IRQ signal is not signaled to the PE. |

If System register access is enabled for EL3 and [ICC_SRE_EL3.DIB](#) == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

FIQBypDisGrp0, bit [5]

When the signaling of FIQs by the CPU interface is disabled, this field partly controls whether the bypass FIQ signal is signaled to the PE for Group 0:

| FIQBypDisGrp0 | Meaning |
|----------------------|--|
| 0b0 | The bypass FIQ signal is signaled to the PE. |
| 0b1 | The bypass FIQ signal is not signaled to the PE. |

If System register access is enabled for EL3 and [ICC_SRE_EL3.DIB](#) == 1, this field is RAO/WI.

If System register access is enabled for EL1, this field is ignored.

If an implementation does not support legacy interrupts, this bit is permitted to be RAO/WI.

For more information, see 'Interrupt bypass support' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

CBPR, bit [4]

Controls whether [GICC_BPR](#) provides common control of preemption to Group 0 and Group 1 interrupts:

| CBPR | Meaning |
|------|--|
| 0b0 | GICC_BPR determines preemption for Group 0 interrupts only. |
| 0b1 | GICC_ABPR determines preemption for Group 1 interrupts. GICC_BPR determines preemption for both Group 0 and Group 1 interrupts. |

This field is an alias of [ICC_CTLR_EL3.CBPR_EL1NS](#).

In a GIC that supports two Security states, when CBPR == 1:

- A Non-secure read of [GICC_BPR](#) returns the value of Secure [GICC_BPR](#).Binary_Point, incremented by 1, and saturated to 0b111.
- Non-secure writes of [GICC_BPR](#) are ignored.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

FIQEn, bit [3]

Controls whether the CPU interface signals Group 0 interrupts to a target PE using the FIQ or IRQ signal:

| FIQEn | Meaning |
|-------|---|
| 0b0 | Group 0 interrupts are signaled using the IRQ signal. |
| 0b1 | Group 0 interrupts are signaled using the FIQ signal. |

Group 1 interrupts are signaled using the IRQ signal only.

If an implementation supports two Security states, this bit is permitted to be RAO/WI.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Bit [2]

Reserved, RES0.

EnableGrp1, bit [1]

This Non-secure field enables the signaling of Group 1 interrupts by the CPU interface to a target PE:

| EnableGrp1 | Meaning |
|------------|--|
| 0b0 | Group 1 interrupt signaling is disabled. |
| 0b1 | Group 1 interrupt signaling is enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

EnableGrp0, bit [0]

Enables the signaling of Group 0 interrupts by the CPU interface to a target PE:

| EnableGrp0 | Meaning |
|------------|--|
| 0b0 | Group 0 interrupt signaling is disabled. |
| 0b1 | Group 0 interrupt signaling is enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing GICC_CTLR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_CTLR](#) and [ICC_MCTLR](#) provide equivalent functionality.
- For AArch64 implementations, [ICC_CTLR_EL1](#) and [ICC_CTLR_EL3](#) provide equivalent functionality.

GICC_CTLR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|--------|-----------|
| GIC CPU interface | 0x0000 | GICC_CTLR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_DIR, CPU Interface Deactivate Interrupt Register

The GICC_DIR characteristics are:

Purpose

When interrupt priority drop is separated from interrupt deactivation, a write to this register deactivates the specified interrupt.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_DIR are RES0.

Attributes

GICC_DIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Accessing GICC_DIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_DIR](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_DIR_EL1](#) provides equivalent functionality.

Writes to this register have an effect only in the following cases:

- When [GICD_CTLR](#).DS == 1, if [GICC_CTLR](#).EOImode == 1.
- In GIC implementations that support two Security states:
 - If the access is Secure and [GICC_CTLR](#).EOImodeS == 1.
 - If the access is Non-secure and [GICC_CTLR](#).EOImodeNS == 1.

The following writes must be ignored:

- Writes to this register when the corresponding EOImode field in [GICC_CTLR](#) == 0. In systems that support system error generation, an implementation might generate a system error.
- Writes to this register when the corresponding EOImode field in [GICC_CTLR](#) == 0 and the corresponding interrupt is not active. In systems that support system error generation, an implementation might generate a system error. In implementations using the GIC Stream Protocol Interface, these writes correspond to a Deactivate packet for an interrupt that is not active. For more information, see 'Deactivate (ICC)' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

If the corresponding EOImode field in [GICC_CTLR](#) is 1 and this register is written to without a corresponding write to [GICC_EOIR](#) or [GICC_AEOIR](#), the interrupt is deactivated but the bit corresponding to it in the active priorities registers remains set.

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_DIR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|--------|----------|
| GIC CPU interface | 0x1000 | GICC_DIR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **WO**.
- When an access is Secure accesses to this register are **WO**.
- When an access is Non-secure accesses to this register are **WO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_EOIR, CPU Interface End Of Interrupt Register

The GICC_EOIR characteristics are:

Purpose

A write to this register performs priority drop for the specified interrupt and, if the appropriate [GICC_CTLR](#).EOImodeS or [GICC_CTLR](#).EOImodeNS field == 0, also deactivates the interrupt.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_EOIR are RES0.

If [GICD_CTLR](#).DS==0:

- This register is Common.
- [GICC_AEOIR](#) is an alias of the Non-secure view of this register.

For Secure writes when [GICD_CTLR](#).DS==0, or for Secure and Non-secure writes when [GICD_CTLR](#).DS==1, the register provides functionality for Group 0 interrupts.

For Non-secure writes when [GICD_CTLR](#).DS==1, the register provides functionality for Group 1 interrupts.

Attributes

GICC_EOIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

For every read of a valid INTID from [GICC_IAR](#), the connected PE must perform a matching write to GICC_EOIR. The value written to GICC_EOIR must be the INTID from [GICC_IAR](#). Reads of INTIDs 1020-1023 do not require matching writes.

Note

Arm recommends that software preserves the entire register value read from [GICC_IAR](#), and writes that value back to GICC_EOIR on completion of interrupt processing.

For nested interrupts, the order of writes to this register must be the reverse of the order of interrupt acknowledgement. Behavior is UNPREDICTABLE if:

- This ordering constraint is not maintained.
- The value written to this register does not match an active interrupt, or the ID of a spurious interrupt.
- The value written to this register does not match the last valid interrupt value read from [GICC_IAR](#).

For general information about the effect of writes to end of interrupt registers, and about the possible separation of the priority drop and interrupt deactivate operations, see 'Interrupt lifecycle' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

If [GICD_CTLR.DS](#)==0:

- [GICC_CTLR.EOImodeS](#) controls the behavior of Secure accesses to GICC_EOIR and [GICC_AEOIR](#).
- [GICC_CTLR.EOImodeNS](#) controls the behavior of Non-secure accesses to GICC_EOIR and [GICC_AEOIR](#).

Accessing GICC_EOIR

The following writes must be ignored:

- Writes of INTIDs 1020-1023.
- Secure writes corresponding to Group 1 interrupts. In systems that support system error generation, an implementation might generate a system error. In this case, GIC behavior is predictable, and the highest Secure active priority (in the Secure copy of [GICC_APR<n>](#)) will be reset if the highest active priority is Secure. System behavior is UNPREDICTABLE.
- Non-secure writes corresponding to Group 0 interrupts when [GICC_CTLR.EOImodeS](#) == 1. In systems that support system error generation, an implementation might generate a system error. In this case, GIC behavior is predictable, and the highest Non-secure active priority (in the Non-secure copy of [GICC_APR<n>](#)) will be reset if the highest active priority is Non-secure. System behavior is UNPREDICTABLE.

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_EOIR0](#) and [ICC_EOIR1](#) provide equivalent functionality.
- For AArch64 implementations, [ICC_EOIR0_EL1](#) and [ICC_EOIR1_EL1](#) provide equivalent functionality.

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_EOIR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|--------|-----------|
| GIC CPU interface | 0x0010 | GICC_EOIR |

This interface is accessible as follows:

- When [GICD_CTLR.DS](#) == 0 accesses to this register are **WO**.
- When an access is Secure accesses to this register are **WO**.
- When an access is Non-secure accesses to this register are **WO**.

GICC_HPPIR, CPU Interface Highest Priority Pending Interrupt Register

The GICC_HPPIR characteristics are:

Purpose

Provides the INTID of the highest priority pending interrupt on the CPU interface.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_HPPIR are RES0.

If [GICD_CTLR.DS](#)==0:

- This register is Common.
- [GICC_AHPPIR](#) is an alias of the Non-secure view of this register.

Attributes

GICC_HPPIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Accessing GICC_HPPIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_HPPIR0](#) and [ICC_HPPIR1](#) provide equivalent functionality.
- For AArch64 implementations, [ICC_HPPIR0_EL1](#) and [ICC_HPPIR1_EL1](#) provide equivalent functionality.

If the highest priority pending interrupt is in Group 0, a Non-secure read of this register returns the special INTID 1023.

For Secure reads when [GICD_CTLR.DS](#)==0, or for Secure and Non-secure reads when [GICD_CTLR.DS](#)==1, returns the special INTID 1022 if the highest priority pending interrupt is in Group 1.

If no interrupts are in the pending state, a read of this register returns the special INTID 1023.

Interrupt identifiers corresponding to an interrupt group that is not enabled are ignored.

If the highest priority pending interrupt is a direct interrupt that is both individually enabled in the Distributor and part of an interrupt group that is enabled in the Distributor, and the interrupt group is disabled in the CPU interface for this PE, this register returns the special INTID 1023.

For more information about pending interrupts that are not considered when determining the highest priority pending interrupt, see 'Preemption' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_HPPIR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|--------|------------|
| GIC CPU interface | 0x0018 | GICC_HPPIR |

This interface is accessible as follows:

- When [GICD_CTLR.DS](#) == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_IAR, CPU Interface Interrupt Acknowledge Register

The GICC_IAR characteristics are:

Purpose

Provides the INTID of the signaled interrupt. A read of this register by the PE acts as an acknowledge for the interrupt.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_IAR are RES0.

This register is available in all configurations of the GIC. If [GICD_CTLR.DS](#)=0:

- This register is Common.
- [GICC_AIAR](#) is an alias of the Non-secure view of this register.

The format of the INTID is governed by whether affinity routing is enabled for a Security state.

Attributes

GICC_IAR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

INTID, bits [23:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

A read of this register returns the INTID of the highest priority pending interrupt for the CPU interface. The read returns a spurious INTID of 1023 if any of the following apply:

- Forwarding of interrupts by the Distributor to the CPU interface is disabled.
- Signaling of interrupts by the CPU interface to the connected PE is disabled.
- There are no pending interrupts on the CPU interface with sufficient priority for the interface to signal it to the PE.

When the GIC returns a valid INTID to a read of this register it treats the read as an acknowledge of that interrupt. In addition, it changes the interrupt status from pending to active, or to active and pending if the pending state of the interrupt persists. Normally, the pending state of an interrupt persists only if the interrupt is level-sensitive and remains asserted.

For every read of a valid INTID from GICC_IAR, the connected PE must perform a matching write to [GICC_EOIR](#).

Note

- Arm recommends that software preserves the entire register value read from this register, and writes that value back to [GICC_EOIR](#) on completion of interrupt processing.
 - For SPIs, although multiple target PEs might attempt to read this register at any time, only one PE can obtain a valid INTID. For more information, see 'Activation' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).
-

Accessing GICC_IAR

When [GICD_CTLR.DS](#)==1, if the highest priority pending interrupt is in Group 1, the special INTID 1022 is returned.

In GIC implementations that support two Security states, if the highest priority pending interrupt is in Group 0, Non-secure reads return the special INTID 1023.

In GIC implementations that support two Security states, if the highest priority pending interrupt is in Group 1, Secure reads return the special INTID 1022.

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_IAR0](#) and [ICC_IAR1](#) provide equivalent functionality.
- For AArch64 implementations, [ICC_IAR0_EL1](#) and [ICC_IAR1_EL1](#) provide equivalent functionality.

When affinity routing is enabled for a Security state, it is a programming error to use memory-mapped registers to access the GIC.

GICC_IAR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|--------|----------|
| GIC CPU interface | 0x000C | GICC_IAR |

This interface is accessible as follows:

- When [GICD_CTLR.DS](#) == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

GICC_IIDR, CPU Interface Identification Register

The GICC_IIDR characteristics are:

Purpose

Provides information about the implementer and revision of the CPU interface.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_IIDR are RES0.

Attributes

GICC_IIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----------------------|----|----|----|----------|----|----|----|-------------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ProductID | | | | | | | | | | | | Architecture_version | | | | Revision | | | | Implementer | | | | | | | | | | | |

ProductID, bits [31:20]

Product Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Architecture_version, bits [19:16]

The version of the GIC architecture that is implemented.

| Architecture_version | Meaning |
|----------------------|--|
| 0b0001 | GICv1. |
| 0b0010 | GICv2. |
| 0b0011 | FEAT_GICv3 memory-mapped interface supported. Support for the System register interface is discoverable from PE registers ID_PFR1 and ID_AA64PFR0_EL1. |
| 0b0100 | FEAT_GICv4 memory-mapped interface supported. Support for the System register interface is discoverable from PE registers ID_PFR1 and ID_AA64PFR0_EL1. |

Other values are reserved.

Revision, bits [15:12]

Revision number for the CPU interface.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 code of the company that implemented the CPU interface.

- Bits [11:8] are the JEP106 continuation code of the implementer. For an Arm implementation, this field is 0x4.
- Bit [7] is always 0.
- Bits [6:0] are the JEP106 identity code of the implementer. For an Arm implementation, bits [7:0] are therefore 0x3B.

Accessing GICC_IIDR

GICC_IIDR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|--------|-----------|
| GIC CPU interface | 0x00FC | GICC_IIDR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_NSAPR<n>, CPU Interface Non-secure Active Priorities Registers, n = 0 - 3

The GICC_NSAPR<n> characteristics are:

Purpose

Provides information about Group 1 interrupt active priorities.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_NSAPR<n> are RES0.

The contents of these registers are IMPLEMENTATION DEFINED with the one architectural requirement that the value 0x00000000 is consistent with no interrupts being active.

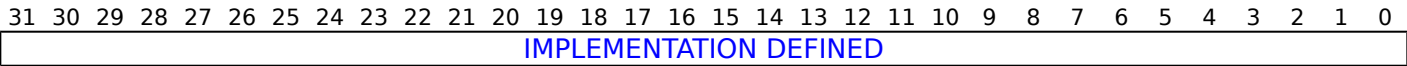
When [GICD_CTLR.DS](#)==0, these registers are RAZ/WI to Non-secure accesses.

GICC_NSAPR1 is only implemented in implementations that support 6 or more bits of priority. GICC_NSAPR2 and GICC_NSAPR3 are only implemented in implementations that support 7 bits of priority.

Attributes

GICC_NSAPR<n> is a 32-bit register.

Field descriptions



IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing GICC_NSAPR<n>

GICC_NSAPR<n> can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|------------------|---------------|
| GIC CPU interface | 0x00E0 + (4 * n) | GICC_NSAPR<n> |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

GICC_PMR, CPU Interface Priority Mask Register

The GICC_PMR characteristics are:

Purpose

This register provides an interrupt priority filter. Only interrupts with a higher priority than the value in this register are signaled to the PE.

Note

Higher interrupt priority corresponds to a lower value of the Priority field.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_PMR are RES0.

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states this register is Common.

Attributes

GICC_PMR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|----------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | Priority | | | | | | | |

Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The priority mask level for the CPU interface. If the priority of the interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

If the GIC implementation supports fewer than 256 priority levels some bits might be RAZ/WI, as follows:

- For 128 supported levels, bit [0] = 0b0.
- For 64 supported levels, bits [1:0] = 0b00.
- For 32 supported levels, bits [2:0] = 0b000.
- For 16 supported levels, bits [3:0] = 0b0000.

For more information, see 'Interrupt prioritization' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GICC_PMR

If the GIC implementation supports two Security states:

- Non-secure accesses to this register can only read or write values corresponding to the lower half of the priority range.
- If a Secure write has programmed the register with a value that corresponds to a value in the upper half of the priority range then:
 - Any Non-secure read of the register returns 0x00, regardless of the value held in the register.
 - Non-secure writes are ignored.

For more information, see 'Interrupt prioritization' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

GICC_PMR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|--------|----------|
| GIC CPU interface | 0x0004 | GICC_PMR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_RPR, CPU Interface Running Priority Register

The GICC_RPR characteristics are:

Purpose

This register indicates the running priority of the CPU interface.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_RPR are RES0.

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states this register is Common.

Attributes

GICC_RPR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|----------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | Priority | | | | | | | |

Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The current running priority on the CPU interface. This is the group priority of the current active interrupt.

If there are no active interrupts on the CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

The priority returned is the group priority as if the BPR was set to the minimum value.

Accessing GICC_RPR

If there is no active interrupt on the CPU interface, the idle priority value is returned.

If the GIC implementation supports two Security states, a Non-secure read of the Priority field returns:

- 0x00 if the field value is less than 0x80.
- The Non-secure view of the Priority value if the field value is 0x80 or more.

For more information, see 'Interrupt prioritization' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Note

Software cannot determine the number of implemented priority bits from this register.

GICC_RPR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|--------|----------|
| GIC CPU interface | 0x0014 | GICC_RPR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICC_STATUSR, CPU Interface Status Register

The GICC_STATUSR characteristics are:

Purpose

Provides software with a mechanism to detect:

- Accesses to reserved locations.
- Writes to read-only locations.
- Reads of write-only locations.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented. Otherwise, direct accesses to GICC_STATUSR are RES0.

If the GIC implementation supports two Security states this register is Banked to provide Secure and Non-secure copies.

This register is used only when System register access is not enabled. If System register access is enabled, this register is not updated. Equivalent functionality might be provided by appropriate traps and exceptions.

Attributes

GICC_STATUSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|---|---|---|-----|---|------|---|------|---|-----|--|-----|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | |
| | | | | | | | | | | | | | RES0 | | | | | | | | | | | | ASV | | WROD | | RWOD | | WRD | | RRD | |

Bits [31:5]

Reserved, RES0.

ASV, bit [4]

Attempted security violation.

| ASV | Meaning |
|-----|---|
| 0b0 | Normal operation. |
| 0b1 | A Non-secure access to a Secure register has been detected. |

Note

This bit is not set to 1 for registers where any of the fields are Non-secure.

WROD, bit [3]

Write to an RO location.

| WROD | Meaning |
|------|--|
| 0b0 | Normal operation. |
| 0b1 | A write to an RO location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

RWOD, bit [2]

Read of a WO location.

| RWOD | Meaning |
|-------------|--|
| 0b0 | Normal operation. |
| 0b1 | A read of a WO location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

WRD, bit [1]

Write to a reserved location.

| WRD | Meaning |
|------------|---|
| 0b0 | Normal operation. |
| 0b1 | A write to a reserved location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

RRD, bit [0]

Read of a reserved location.

| RRD | Meaning |
|------------|--|
| 0b0 | Normal operation. |
| 0b1 | A read of a reserved location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

Accessing GICC_STATUSR

This is an optional register. If the register is not implemented, the location is RAZ/WI.

If this register is implemented, [GICV_STATUSR](#) must also be implemented.

GICC_STATUSR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------|---------------|------------------|
| GIC CPU interface | 0x002C | GICC_STATUSR (S) |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.

| Component | Offset | Instance |
|-------------------|---------------|-------------------|
| GIC CPU interface | 0x002C | GICC_STATUSR (NS) |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

GICD_CLRSPI_NSR, Clear Non-secure SPI Pending Register

The GICD_CLRSPI_NSR characteristics are:

Purpose

Removes the pending state from a valid SPI if permitted by the Security state of the access and the [GICD_NSACR<n>](#) value for that SPI.

A write to this register changes the state of a pending SPI to inactive, and the state of an active and pending SPI to active.

Configuration

If [GICD_TYPER](#).MBIS == 0, this register is reserved.

When [GICD_CTLR](#).DS == 1, this register provides functionality for all SPIs.

Attributes

GICD_CLRSPI_NSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | |

Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

The INTID of the SPI.

The function of this register depends on whether the targeted SPI is configured to be an edge-triggered or level-sensitive interrupt:

- For an edge-triggered interrupt, a write to [GICD_SETSPI_NSR](#) or [GICD_SETSPI_SR](#) adds the pending state to the targeted interrupt. It will stop being pending on activation, or if the pending state is removed by a write to [GICD_CLRSPI_NSR](#), [GICD_CLRSPI_SR](#), or [GICD_ICPENDR<n>](#).
- For a level-sensitive interrupt, a write to [GICD_SETSPI_NSR](#) or [GICD_SETSPI_SR](#) adds the pending state to the targeted interrupt. It will remain pending until it is deasserted by a write to [GICD_CLRSPI_NSR](#) or [GICD_CLRSPI_SR](#). If the interrupt is activated between having the pending state added and being deactivated, then the interrupt will be active and pending.

Accessing GICD_CLRSPI_NSR

Writes to this register have no effect if:

- The value written specifies a Secure SPI, the value is written by a Non-secure access, and the value of the corresponding [GICD_NSACR<n>](#) register is less than 0b10.
- The value written specifies an invalid SPI.
- The SPI is not pending.

16-bit accesses to bits [15:0] of this register must be supported.

Note

A Secure access to this register can clear the pending state of any valid SPI.

GICD_CLRSPI_NSR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|--------|-----------------|
| GIC Distributor | Dist_base | 0x0048 | GICD_CLRSPI_NSR |

Accesses on this interface are **WO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_CLRSPI_SR, Clear Secure SPI Pending Register

The GICD_CLRSPI_SR characteristics are:

Purpose

Removes the pending state from a valid SPI.

A write to this register changes the state of a pending SPI to inactive, and the state of an active and pending SPI to active.

Configuration

If [GICD_TYPER](#).MBIS == 0, this register is reserved.

When [GICD_CTLR](#).DS == 1, this register is WI.

Attributes

GICD_CLRSPI_SR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | |

Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

The INTID of the SPI.

The function of this register depends on whether the targeted SPI is configured to be an edge-triggered or level-sensitive interrupt:

- For an edge-triggered interrupt, a write to [GICD_SETSPI_NSR](#) or [GICD_SETSPI_SR](#) adds the pending state to the targeted interrupt. It will stop being pending on activation, or if the pending state is removed by a write to [GICD_CLRSPI_NSR](#), [GICD_CLRSPI_SR](#), or [GICD_ICPENDR<n>](#).
- For a level-sensitive interrupt, a write to [GICD_SETSPI_NSR](#) or [GICD_SETSPI_SR](#) adds the pending state to the targeted interrupt. It will remain pending until it is deasserted by a write to [GICD_CLRSPI_NSR](#) or [GICD_CLRSPI_SR](#). If the interrupt is activated between having the pending state added and being deactivated, then the interrupt will be active and pending.

Accessing GICD_CLRSPI_SR

Writes to this register have no effect if:

- The value is written by a Non-secure access.
- The value written specifies an invalid SPI.
- The SPI is not pending.

16-bit accesses to bits [15:0] of this register must be supported.

GICD_CLRSPI_SR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|--------|----------------|
| GIC Distributor | Dist_base | 0x0058 | GICD_CLRSPI_SR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 1 accesses to this register are **WI**.
- When GICD_CTLR.DS == 0 and an access is Secure accesses to this register are **WO**.
- When GICD_CTLR.DS == 0 and an access is Non-secure accesses to this register are **WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRoot() accesses to this register are **WO**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRealm() accesses to this register are **WI**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_CPENDSGIR<n>, SGI Clear-Pending Registers, n = 0 - 3

The GICD_CPENDSGIR<n> characteristics are:

Purpose

Removes the pending state from an SGI.

A write to this register changes the state of a pending SGI to inactive, and the state of an active and pending SGI to active.

Configuration

Four SGI clear-pending registers are implemented. Each register contains eight clear-pending bits for each of four SGIs, for a total of 16 possible SGIs.

In multiprocessor implementations, each PE has a copy of these registers.

Attributes

GICD_CPENDSGIR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|----|----|----|----|----|----|----|---|----|----|----|----|----|----|----|---|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SGI_clear_pending_bits3 | | | | | | | | SGI_clear_pending_bits2 | | | | | | | | SGI_clear_pending_bits1 | | | | | | | | SGI_clear_pending_bits0 | | | | | | | |

SGI_clear_pending_bits<x>, bits [8x+7:8x], for x = 3 to 0

Removes the pending state from SGI number $4n + x$ for the PE corresponding to the bit number written to.

Reads and writes have the following behavior:

| SGI_clear_pending_bits<x> | Meaning |
|---------------------------|--|
| 0x00 | If read, indicates that the SGI from the corresponding PE is not pending and is not active and pending. If written, has no effect. |
| 0x01 | If read, indicates that the SGI from the corresponding PE is pending or is active and pending. If written, removes the pending state from the SGI for the corresponding PE. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For SGI ID m , generated by processing element C writing to the corresponding [GICD_SGIR](#) field, where DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_CPENDSGIR<n> number is given by $n = m \text{ DIV } 4$.
- The offset of the required register is $(0xF10 + (4n))$.
- The offset of the required field within the register GICD_CPENDSGIR<n> is given by $m \text{ MOD } 4$.
- The required bit in the 8-bit SGI clear-pending field m is bit C .

Accessing GICD_CPENDSGIR<n>

These registers are used only when affinity routing is not enabled. When affinity routing is enabled, this register is RES0. An implementation is permitted to make the register RAZ/WI in this case.

A register bit that corresponds to an unimplemented SGI is RAZ/WI.

These registers are byte-accessible.

If the GIC implementation supports two Security states:

- A register bit that corresponds to a Group 0 interrupt is RAZ/WI to Non-secure accesses.
- Register bits corresponding to unimplemented PEs are RAZ/WI.

GICD_CPENDSGIR<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|-------------------|
| GIC Distributor | Dist_base | 0x0F10 + (4 * n) | GICD_CPENDSGIR<n> |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_CTLR, Distributor Control Register

The GICD_CTLR characteristics are:

Purpose

Enables interrupts and affinity routing.

Configuration

The format of this register depends on the Security state of the access and the number of Security states supported, which is specified by GICD_CTLR.DS.

Attributes

GICD_CTLR is a 32-bit register.

Field descriptions

When access is Secure, in a system that supports two Security states:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|----|--------|-------|------|-------------|--------------|--------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RWP | | | | | | | | | | | | | | | | | | | | | | | | E1NWF | DS | ARE_NS | ARE_S | RES0 | EnableGrp1S | EnableGrp1NS | EnableGrp1NS |

RWP, bit [31]

Register Write Pending. Read only. Indicates whether a register write is in progress or not:

| RWP | Meaning |
|------------|---|
| 0b0 | No register write in progress. The effects of previous register writes to the affected register fields are visible to all logical components of the GIC architecture, including the CPU interfaces. |
| 0b1 | Register write in progress. The effects of previous register writes to the affected register fields are not guaranteed to be visible to all logical components of the GIC architecture, including the CPU interfaces, as the effects of the changes are still being propagated. |

This field tracks writes to:

- GICD_CTLR[2:0], the Group Enables, for transitions from 1 to 0 only.
- GICD_CTLR[7:4], the ARE bits, E1NWF bit and DS bit.
- GICD_ICENABLER<n>.

Updates to other register fields are not tracked by this field.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [30:8]

Reserved, RES0.

E1NWF, bit [7]

Enable 1 of N Wakeup Functionality.

It is IMPLEMENTATION DEFINED whether this bit is programmable, or RAZ/WI.

If it is implemented, then it has the following behavior:

| E1NWF | Meaning |
|--------------|---|
| 0b0 | A PE that is asleep cannot be picked for 1 of N interrupts. |
| 0b1 | A PE that is asleep can be picked for 1 of N interrupts as determined by IMPLEMENTATION DEFINED controls. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

DS, bit [6]

Disable Security.

| DS | Meaning |
|-----------|---|
| 0b0 | Non-secure accesses are not permitted to access and modify registers that control Group 0 interrupts. |
| 0b1 | Non-secure accesses are permitted to access and modify registers that control Group 0 interrupts. |

If DS is written from 0 to 1 when GICD_CTLR.ARE_S == 1, then GICD_CTLR.ARE for the single Security state is RAO/WI.

If the Distributor only supports a single Security state, this bit is RAO/WI.

If the Distributor supports two Security states, it IMPLEMENTATION DEFINED whether this bit is programmable or implemented as RAZ/WI.

When this field is set to 1, all accesses to GICD_CTLR access the single Security state view, and all bits are accessible.

When set to 1, this field can only be cleared by a hardware reset.

Writing this bit from 0 to 1 is UNPREDICTABLE if any of the following is true:

- [GICD_CTLR.EnableGrp0](#)==1.
- [GICD_CTLR.EnableGrp1S](#)==1.
- [GICD_CTLR.EnableGrp1NS](#)==1.
- One or more INTID is in the Active or Active and Pending state.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

ARE_NS, bit [5]

Affinity Routing Enable, Non-secure state.

| ARE_NS | Meaning |
|---------------|---|
| 0b0 | Affinity routing disabled for Non-secure state. |
| 0b1 | Affinity routing enabled for Non-secure state. |

When affinity routing is enabled for the Secure state, this field is RAO/WI.

Changing the ARE_NS settings from 0 to 1 is UNPREDICTABLE except when GICD_CTLR.EnableGrp1 Non-secure == 0.

Changing the ARE_NS settings from 1 to 0 is UNPREDICTABLE.

If GICv2 backwards compatibility for Non-secure state is not implemented, this field is RAO/WI.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

ARE_S, bit [4]

Affinity Routing Enable, Secure state.

| ARE_S | Meaning |
|--------------|---|
| 0b0 | Affinity routing disabled for Secure state. |
| 0b1 | Affinity routing enabled for Secure state. |

Changing the ARE_S setting from 0 to 1 is UNPREDICTABLE except when all of the following apply:

- GICD_CTLR.EnableGrp0==0.
- GICD_CTLR.EnableGrp1S==0.
- GICD_CTLR.EnableGrp1NS==0.

Changing the ARE_S settings from 1 to 0 is UNPREDICTABLE.

If GICv2 backwards compatibility for Secure state is not implemented, this field is RAO/WI.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Bit [3]

Reserved, RES0.

EnableGrp1S, bit [2]

Enable Secure Group 1 interrupts.

| EnableGrp1S | Meaning |
|--------------------|---|
| 0b0 | Secure Group 1 interrupts are disabled. |
| 0b1 | Secure Group 1 interrupts are enabled. |

If GICD_CTLR.ARE_S == 0, this field is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

EnableGrp1NS, bit [1]

Enable Non-secure Group 1 interrupts.

| EnableGrp1NS | Meaning |
|---------------------|---|
| 0b0 | Non-secure Group 1 interrupts are disabled. |
| 0b1 | Non-secure Group 1 interrupts are enabled. |

Note

This field also controls whether LPIs are forwarded to the PE.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

EnableGrp0, bit [0]

Enable Group 0 interrupts.

| EnableGrp0 | Meaning |
|-------------------|----------------------------------|
| 0b0 | Group 0 interrupts are disabled. |
| 0b1 | Group 0 interrupts are enabled. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

When access is Non-secure, in a system that supports two Security states:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|------|-------------|----|------------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RWP | RES0 | | | | | | | | | | | | | | | | ARE_NS | | RES0 | EnableGrp1A | | EnableGrp1 | | | | | | | | | |

RWP, bit [31]

This bit is a read-only alias of the Secure GICD_CTLR.RWP bit.

Bits [30:5]

Reserved, RES0.

ARE_NS, bit [4]

This bit is a read-write alias of the Secure GICD_CTLR.ARE_NS bit.

If GICv2 backwards compatibility for Non-secure state is not implemented, this field is RAO/WI.

Bits [3:2]

Reserved, RES0.

EnableGrp1A, bit [1]

If ARE_NS == 1, then this bit is a read-write alias of the Secure GICD_CTLR.EnableGrp1NS bit.

If ARE_NS == 0, then this bit is RES0.

EnableGrp1, bit [0]

If ARE_NS == 0, then this bit is a read-write alias of the Secure GICD_CTLR.EnableGrp1NS bit.

If ARE_NS == 1, then this bit is RES0.

When in a system that supports only a single Security state:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|-------|----|------|-----|------|------------|---|------------|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RWP | RES0 | | | | | | | | | | | | | | | | nASSGReq | | E1NWF | DS | RES0 | ARE | RES0 | EnableGrp1 | | EnableGrp0 | | | | | |

RWP, bit [31]

Register Write Pending. Read only. Indicates whether a register write is in progress or not:

| RWP | Meaning |
|-----|---|
| 0b0 | No register write in progress. The effects of previous register writes to the affected register fields are visible to all logical components of the GIC architecture, including the CPU interfaces. |
| 0b1 | Register write in progress. The effects of previous register writes to the affected register fields are not guaranteed to be visible to all logical components of the GIC architecture, including the CPU interfaces, as the effects of the changes are still being propagated. |

This field tracks updates to:

- GICD_CTLR[2:0], the Group Enables, for transitions from 1 to 0 only.
- GICD_CTLR[7:4], the ARE bits, E1NWF bit and DS bit.

- GICD_ICENABLER<n>, the bits that allow disabling of SPIs.

Updates to other register fields are not tracked by this field.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [30:9]

Reserved, RES0.

nASSGImeq, bit [8]

When FEAT_GICv4p1 is implemented:

Controls whether SGIs have an active state.

This bit is RES0 if [GICD_TYPER2](#).GICD_TYPER2.nASSGImeq is 0.

This bit is WI when any of GICD_CTLR.{EnableGrp0,EnableGrp1} is 1.

| nASSGImeq | Meaning |
|-----------|---|
| 0b0 | SGIs have an active state and must be deactivated. |
| 0b1 | SGIs do not have an active state and do not require deactivation. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Otherwise:

Reserved, RES0.

E1NWF, bit [7]

Enable 1 of N Wakeup Functionality.

It is IMPLEMENTATION DEFINED whether this bit is programmable, or RAZ/WI.

If it is implemented, then it has the following behavior:

| E1NWF | Meaning |
|-------|---|
| 0b0 | A PE that is asleep cannot be picked for 1 of N interrupts. |
| 0b1 | A PE that is asleep can be picked for 1 of N interrupts as determined by IMPLEMENTATION DEFINED controls. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

DS, bit [6]

Disable Security. This field is RAO/WI.

Bit [5]

Reserved, RES0.

ARE, bit [4]

Affinity Routing Enable.

| ARE | Meaning |
|-----|----------------------------|
| 0b0 | Affinity routing disabled. |
| 0b1 | Affinity routing enabled. |

Changing the ARE settings from 0 to 1 is UNPREDICTABLE except when all of the following apply:

- GICD_CTLR.EnableGrp1==0.
- GICD_CTLR.EnableGrp0==0.

Changing ARE from 1 to 0 is UNPREDICTABLE.

If GICv2 backwards compatibility is not implemented, this field is RAO/WI.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Bits [3:2]

Reserved, RES0.

EnableGrp1, bit [1]

Enable Group 1 interrupts.

| EnableGrp1 | Meaning |
|------------|------------------------------|
| 0b0 | Group 1 interrupts disabled. |
| 0b1 | Group 1 interrupts enabled. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

EnableGrp0, bit [0]

Enable Group 0 interrupts.

| EnableGrp0 | Meaning |
|------------|----------------------------------|
| 0b0 | Group 0 interrupts are disabled. |
| 0b1 | Group 0 interrupts are enabled. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICD_CTLR

If an interrupt is pending within a CPU interface when the corresponding GICD_CTLR.EnableGrpX bit is written from 1 to 0 the interrupt must be retrieved from the CPU interface.

Note

This might have no effect on the forwarded interrupt if it has already been activated. When a write changes the value of ARE for a Security state or the value of the DS bit, the format used for interpreting the remaining bits provided in the write data is the format that applied before the write takes effect.

GICD_CTLR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|--------|-----------|
| GIC Distributor | Dist_base | 0x0000 | GICD_CTLR |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICACTIVER<n>, Interrupt Clear-Active Registers, n = 0 - 31

The GICD_ICACTIVER<n> characteristics are:

Purpose

Deactivates the corresponding interrupt. These registers are used when saving and restoring GIC state.

Configuration

These registers are available in all GIC configurations. If [GICD_CTLR.DS](#)==0, these registers are Common.

The number of implemented GICD_ICACTIVER<n> registers is ([GICD_TYPER.ITLinesNumber](#)+1). Registers are numbered from 0.

GICD_ICACTIVER0 is Banked for each connected PE with [GICR_TYPER.Processor_Number](#) < 8.

Accessing GICD_ICACTIVER0 from a PE with [GICR_TYPER.Processor_Number](#) > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ICACTIVER<n> is a 32-bit register.

Field descriptions

| | | | | | | |
|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | |
| Clear_active_bit31 | Clear_active_bit30 | Clear_active_bit29 | Clear_active_bit28 | Clear_active_bit27 | Clear_active_bit26 | Clear_active_bit25 |

Clear_active_bit<x>, bit [x], for x = 31 to 0

Removes the active state from interrupt number 32n + x. Reads and writes have the following behavior:

| Clear_active_bit<x> | Meaning |
|---------------------|--|
| 0b0 | If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is active, or is active and pending. If written, deactivates the corresponding interrupt, if the interrupt is active. If the interrupt is already deactivated, the write has no effect. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICACTIVER<n> number, n, is given by n = m DIV 32.
- The offset of the required GICD_ICACTIVER is (0x380 + (4*n)).
- The bit number of the required group modifier bit in this register is m MOD 32.

Accessing GICD_ICACTIVER<n>

When affinity routing is enabled for the Security state of an interrupt, the bits corresponding to SGIs and PPIs in that Security state are RAZ/WI, and equivalent functionality for SGIs and PPIs is provided by [GICR_ICACTIVER0](#).

Bits corresponding to unimplemented interrupts are RAZ/WI.

If [GICD_CTLR.DS](#)==0, unless the [GICD_NSACR<n>](#) registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

GICD_ICACTIVER<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------------|-------------------|
| GIC Distributor | Dist_base | 0x0380 + (4 * n) | GICD_ICACTIVER<n> |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICACTIVER<n>E, Interrupt Clear-Active Registers (extended SPI range), n = 0 - 31

The GICD_ICACTIVER<n>E characteristics are:

Purpose

Removes the active state from the corresponding SPI in the extended SPI range.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICD_ICACTIVER<n>E are RES0.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ICACTIVER<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.

Attributes

GICD_ICACTIVER<n>E is a 32-bit register.

Field descriptions

| | | | | | | |
|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|-----|
| 31 | 30 | 29 | 28 | 27 | 26 | Cle |
| Clear_active_bit31 | Clear_active_bit30 | Clear_active_bit29 | Clear_active_bit28 | Clear_active_bit27 | Clear_active_bit26 | Cle |

Clear_active_bit<x>, bit [x], for x = 31 to 0

For the extended SPIs, removes the active state to interrupt number x. Reads and writes have the following behavior:

| Clear_active_bit<x> | Meaning |
|---------------------|--|
| 0b0 | If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is active, or is active and pending. If written, deactivates the corresponding interrupt, if the interrupt is active. If the interrupt is already deactivated, the write has no effect. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICACTIVER<n>E number, n, is given by $n = (m-4096) \text{ DIV } 32$.
- The offset of the required GICD_ICACTIVER<n>E is $(0 \times 1C00 + (4 \times n))$.
- The bit number of the required group modifier bit in this register is $(m-4096) \text{ MOD } 32$.

Accessing GICD_ICACTIVER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ICACTIVER<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_ICACTIVER<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|--------------------|
| GIC Distributor | Dist_base | 0x1C00 + (4 * n) | GICD_ICACTIVER<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICENABLER<n>, Interrupt Clear-Enable Registers, n = 0 - 31

The GICD_ICENABLER<n> characteristics are:

Purpose

Disables forwarding of the corresponding interrupt to the CPU interfaces.

Configuration

These registers are available in all GIC configurations. If [GICD_CTLR.DS](#)=0, these registers are Common.

The number of implemented [GICD_ICENABLER<n>](#) registers is ([GICD_TYPER.ITLinesNumber](#)+1). Registers are numbered from 0.

GICD_ICENABLER0 is Banked for each connected PE with [GICR_TYPER.Processor_Number](#) < 8.

Accessing GICD_ICENABLER0 from a PE with [GICR_TYPER.Processor_Number](#) > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ICENABLER<n> is a 32-bit register.

Field descriptions

| | | | | | |
|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 |
| Clear_enable_bit31 | Clear_enable_bit30 | Clear_enable_bit29 | Clear_enable_bit28 | Clear_enable_bit27 | Clear_enable_bit26 |

Clear_enable_bit<x>, bit [x], for x = 31 to 0

For SPIs and PPIs, controls the forwarding of interrupt number $32n + x$ to the CPU interfaces. Reads and writes have the following behavior:

| Clear_enable_bit<x> | Meaning |
|---------------------|--|
| 0b0 | If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect. |
| 0b1 | If read, indicates that forwarding of the corresponding interrupt is enabled. If written, disables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 0. |

For SGIs, the behavior of this bit is IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICENABLER<n> number, n, is given by $n = m \text{ DIV } 32$.
- The offset of the required GICD_ICENABLER is $(0 \times 180 + (4 * n))$.
- The bit number of the required group modifier bit in this register is $m \text{ MOD } 32$.

Note

Writing a 1 to a GICD_ICENABLER<n> bit only disables the forwarding of the corresponding interrupt from the Distributor to any CPU interface. It does not prevent the interrupt from changing state, for example becoming pending or active and pending if it is already active.

Accessing GICD_ICENABLER<n>

For SGIs and PPIs:

- When ARE is 1 for the Security state of an interrupt, the field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by GICR_ICENABLER0.

Bits corresponding to unimplemented interrupts are RAZ/WI.

When [GICD_CTLR.DS](#)=0, bits corresponding to Group 0 and Secure Group 1 interrupts are RAZ/WI to Non-secure accesses.

It is IMPLEMENTATION DEFINED whether implemented SGIs are permanently enabled, or can be enabled and disabled by writes to [GICD_ISENABLER<n>](#) and [GICD_ICENABLER<n>](#) where n=0.

Completion of a write to this register does not guarantee that the effects of the write are visible throughout the affinity hierarchy. To ensure an enable has been cleared, software must write to the register with bits set to 1 to clear the required enables. Software must then poll [GICD_CTLR.RWP](#) until it has the value zero.

GICD_ICENABLER<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------------|-------------------|
| GIC Distributor | Dist_base | 0x0180 + (4 * n) | GICD_ICENABLER<n> |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICENABLER<n>E, Interrupt Clear-Enable Registers, n = 0 - 31

The GICD_ICENABLER<n>E characteristics are:

Purpose

Disables forwarding of the corresponding SPI in the extended SPI range to the CPU interfaces.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICD_ICENABLER<n>E are RES0.

When [GICD_TYPER.ESPI](#)==0, these registers are RES0.

When [GICD_TYPER.ESPI](#)==1, the number of implemented [GICD_ICENABLER<n>E](#) registers is ([GICD_TYPER.ESPI_range](#)+1). Registers are numbered from 0.

Attributes

GICD_ICENABLER<n>E is a 32-bit register.

Field descriptions

| | | | | | |
|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 |
| Clear_enable_bit31 | Clear_enable_bit30 | Clear_enable_bit29 | Clear_enable_bit28 | Clear_enable_bit27 | Clear_enable_bit26 |

Clear_enable_bit<x>, bit [x], for x = 31 to 0

For the extended SPI range, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

| Clear_enable_bit<x> | Meaning |
|---------------------|---|
| 0b0 | If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect. |
| 0b1 | If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 0. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICENABLER<n>E number, n, is given by $n = (m-4096) \text{ DIV } 32$.
- The offset of the required GICD_ICENABLER<n>E is $(0x1400 + (4*n))$.
- The bit number of the required group modifier bit in this register is $(m-4096) \text{ MOD } 32$.

Accessing GICD_ICENABLER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ICENABLER<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_ICENABLER<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------------|--------------------|
| GIC Distributor | Dist_base | 0x1400 + (4 * n) | GICD_ICENABLER<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICFGR<n>, Interrupt Configuration Registers, n = 0 - 63

The GICD_ICFGR<n> characteristics are:

Purpose

Determines whether the corresponding interrupt is edge-triggered or level-sensitive.

Configuration

These registers are available in all GIC configurations. If the GIC implementation supports two Security states, these registers are Common.

GICD_ICFGR1 is Banked for each connected PE with [GICR_TYPER](#).Processor_Number < 8.

Accessing GICD_ICFGR1 from a PE with [GICR_TYPER](#).Processor_Number > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

For SGIs and PPIs:

- When ARE is 1 for the Security state of an interrupt, the field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by GICR_ICFGR<n>

For each supported PPI, it is IMPLEMENTATION DEFINED whether software can program the corresponding Int_config field.

For SGIs, Int_config fields are RO, meaning that GICD_ICFGR0 is RO.

Changing Int_config when the interrupt is individually enabled is UNPREDICTABLE.

Changing the interrupt configuration between level-sensitive and edge-triggered (in either direction) at a time when there is a pending interrupt will leave the interrupt in an UNKNOWN pending state.

Fields corresponding to unimplemented interrupts are RAZ/WI.

Attributes

GICD_ICFGR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | |
|------------------------------|------------------------------|------------------------------|------------------------------|------------------------------|------------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 |
| Int_config15 | Int_config14 | Int_config13 | Int_config12 | Int_config11 | Int_config10 | Int_config9 | Int_config8 | Int_config7 | Int_config6 | Int_config5 | Int_config4 | Int_config3 | Int_config2 | Int_config1 | Int_config0 | | | |

Int_config<x>, bits [2x+1:2x], for x = 15 to 0

Indicates whether the interrupt is level-sensitive or edge-triggered.

| Int_config<x> | Meaning |
|---------------|---|
| 0b00 | Corresponding interrupt is level-sensitive. |
| 0b10 | Corresponding interrupt is edge-triggered. |

Int_config[0] (bit [2x]) is RES0.

For SGIs, this field always indicates edge-triggered.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICD_ICFGR<n>

For SPIs and PPIs, when [GICD_CTLR.DS](#)==0, a register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

GICD_ICFGR<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|---------------|
| GIC Distributor | Dist_base | 0x0C00 + (4 * n) | GICD_ICFGR<n> |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICFGR<n>E, Interrupt Configuration Registers (Extended SPI Range), n = 0 - 63

The GICD_ICFGR<n>E characteristics are:

Purpose

Determines whether the corresponding SPI in the extended SPI range is edge-triggered or level-sensitive.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICD_ICFGR<n>E are RES0.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ICFGR<n>E registers is ((GICD_TYPER.ESPI_range+1)*2). Registers are numbered from 0.

Attributes

GICD_ICFGR<n>E is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | |
|--------------|--------------|--------------|--------------|--------------|--------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 |
| Int_config15 | Int_config14 | Int_config13 | Int_config12 | Int_config11 | Int_config10 | Int_config9 | Int_config8 | Int_config7 | Int_config6 | Int_config5 | Int_config4 | Int_config3 | Int_config2 | Int_config1 | Int_config0 | Int_config0 | Int_config0 | Int_config0 |

Int_config<x>, bits [2x+1:2x], for x = 15 to 0

Indicates whether the interrupt is level-sensitive or edge-triggered.

Int_config[0] (bit[2x]) is RES0.

| Int_config<x> | Meaning |
|---------------|---|
| 0b00 | Corresponding interrupt is level-sensitive. |
| 0b10 | Corresponding interrupt is edge-triggered. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICD_ICFGR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ICFGR<n>E, the corresponding bit is RES0.

When GICD_CTLR.DS==0, a register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_ICFGR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|-------|--------|----------|
|-----------|-------|--------|----------|

| | | | |
|--------------------|-----------|---------------------|----------------|
| GIC Distributor | Dist_base | 0x3000 + (4 * n) | GICD_ICFGR<n>E |
|--------------------|-----------|---------------------|----------------|

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ICPENDR<n>, Interrupt Clear-Pending Registers, n = 0 - 31

The GICD_ICPENDR<n> characteristics are:

Purpose

Removes the pending state from the corresponding interrupt.

Configuration

These registers are available in all GIC configurations. If [GICD_CTLR.DS](#)==0, these registers are Common.

The number of implemented [GICD_ICPENDR<n>](#) registers is ([GICD_TYPER.ITLinesNumber](#)+1). Registers are numbered from 0.

GICD_ICPENDR0 is Banked for each connected PE with [GICR_TYPER.Processor_Number](#) < 8.

Accessing GICD_ICPENDR0 from a PE with [GICR_TYPER.Processor_Number](#) > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ICPENDR<n> is a 32-bit register.

Field descriptions

| 31 | 30 | 29 | 28 | 27 | 26 |
|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| Clear_pending_bit31 | Clear_pending_bit30 | Clear_pending_bit29 | Clear_pending_bit28 | Clear_pending_bit27 | Clear_pending_bit26 |

Clear_pending_bit<x>, bit [x], for x = 31 to 0

For SPIs and PPIs, removes the pending state from interrupt number 32n + x. Reads and writes have the following behavior:

| Clear_pending_bit<x> | Meaning |
|----------------------|---|
| 0b0 | If read, indicates that the corresponding interrupt is not pending on any PE. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is pending, or active and pending. If written, changes the state of the corresponding interrupt from pending to inactive, or from active and pending to active. This has no effect in the following cases: <ul style="list-style-type: none"> If the interrupt is an SGI. In this case, the write is ignored. The pending state of an SGI can be cleared using GICD_CPENDSGIR<n>. If the interrupt is not pending and is not active and pending. If the interrupt is a level-sensitive interrupt that is pending or active and pending for a reason other than a write to GICD_ISPENDR<n>. In this case, if the interrupt signal continues to be asserted, the interrupt remains pending or active and pending. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICPENDR<n> number, n, is given by $n = m \text{ DIV } 32$.
- The offset of the required GICD_ICPENDR is $(0x200 + (4*n))$.
- The bit number of the required group modifier bit in this register is $m \text{ MOD } 32$.

Accessing GICD_ICPENDR<n>

Clear-pending bits for SGIs are RO/WI.

When affinity routing is enabled for the Security state of an interrupt:

- Bits corresponding to SGIs and PPIs are RAZ/WI, and equivalent functionality for SGIs and PPIs is provided by [GICR_ICPENDR0](#).
- Bits corresponding to Group 0 and Group 1 Secure interrupts can only be cleared by Secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

If [GICD_CTLR.DS](#)=0, unless the [GICD_NSACR<n>](#) registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

GICD_ICPENDR<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|-----------------|
| GIC Distributor | Dist_base | 0x0280 + (4 * n) | GICD_ICPENDR<n> |

Accesses on this interface are **RW**.

GICD_ICPENDR<n>E, Interrupt Clear-Pending Registers (extended SPI range), n = 0 - 31

The GICD_ICPENDR<n>E characteristics are:

Purpose

Removes the pending state to the corresponding SPI in the extended SPI range.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICD_ICPENDR<n>E are RES0.

When [GICD_TYPER.ESPI](#)==0, these registers are RES0.

When [GICD_TYPER.ESPI](#)==1, the number of implemented GICD_ICPENDR<n>E registers is ([GICD_TYPER.ESPI_range](#)+1). Registers are numbered from 0.

Attributes

GICD_ICPENDR<n>E is a 32-bit register.

Field descriptions

| | | | | | |
|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 |
| Clear_pending_bit31 | Clear_pending_bit30 | Clear_pending_bit29 | Clear_pending_bit28 | Clear_pending_bit27 | Clear_pending_bit26 |

Clear_pending_bit<x>, bit [x], for x = 31 to 0

For the extended PPIs, removes the pending state to interrupt number x. Reads and writes have the following behavior:

| Clear_pending_bit<x> | Meaning |
|----------------------|---|
| 0b0 | If read, indicates that the corresponding interrupt is not pending. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is pending, or active and pending. If written, changes the state of the corresponding interrupt from pending to inactive, or from active and pending to active. This has no effect in the following cases: <ul style="list-style-type: none">If the interrupt is not pending and is not active and pending.If the interrupt is a level-sensitive interrupt that is pending or active and pending for a reason other than a write to GICD_ICPENDR<n>E. In this case, if the interrupt signal continues to be asserted, the interrupt remains pending or active and pending. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ICPENDR<n>E number, n, is given by $n = (m-4096) \text{ DIV } 32$.
- The offset of the required GICD_ICPENDR<n>E is $(0x1800 + (4*n))$.
- The bit number of the required group modifier bit in this register is $(m-4096) \text{ MOD } 32$.

Accessing GICD_ICPENDR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ICPENDR<n>E, the corresponding bit is RES0.

When [GICD_CTLR](#).DS==0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_ICPENDR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|------------------|
| GIC Distributor | Dist_base | 0x1800 + (4 * n) | GICD_ICPENDR<n>E |

Accesses on this interface are **RW**.

GICD_IGROUPR<n>, Interrupt Group Registers, n = 0 - 31

The GICD_IGROUPR<n> characteristics are:

Purpose

Controls whether the corresponding interrupt is in Group 0 or Group 1.

Configuration

These registers are available in all GIC configurations. If [GICD_CTLR.DS](#)==0, these registers are Secure.

The number of implemented GICD_IGROUPR<n> registers is ([GICD_TYPER.ITLinesNumber](#)+1). Registers are numbered from 0.

GICD_IGROUPR0 is Banked for each connected PE with [GICR_TYPER.Processor_Number](#) < 8.

Accessing GICD_IGROUPR0 from a PE with [GICR_TYPER.Processor_Number](#) > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_IGROUPR<n> is a 32-bit register.

Field descriptions

| | | | | | |
|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 |
| Group_status_bit31 | Group_status_bit30 | Group_status_bit29 | Group_status_bit28 | Group_status_bit27 | Group_status_bit26 |

Group_status_bit<x>, bit [x], for x = 31 to 0

Group status bit.

| Group_status_bit<x> | Meaning |
|---------------------|---|
| 0b0 | When GICD_CTLR.DS ==1, the corresponding interrupt is Group 0. When GICD_CTLR.DS ==0, the corresponding interrupt is Secure. |
| 0b1 | When GICD_CTLR.DS ==1, the corresponding interrupt is Group 1. When GICD_CTLR.DS ==0, the corresponding interrupt is Non-secure Group 1. |

If affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in [GICD_IGRPMODR<n>](#) to form a 2-bit field that defines an interrupt group. The encoding of this field is described in [GICD_IGRPMODR<n>](#).

If affinity routing is disabled for the Security state of an interrupt, then:

- The corresponding [GICD_IGRPMODR<n>](#) bit is RES0.
- For Secure interrupts, the interrupt is Secure Group 0.
- For Non-secure interrupts, the interrupt is Non-secure Group 1.

The reset behavior of this field is:

- On a GIC reset:

- When n == 0, this field resets to an UNKNOWN value.
- When n > 0, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IGROUPR<n> number, n, is given by $n = m \text{ DIV } 32$.
- The offset of the required GICD_IGROUP is $(0x080 + (4*n))$.
- The bit number of the required group modifier bit in this register is $m \text{ MOD } 32$.

Accessing GICD_IGROUPR<n>

For SGIs and PPIs:

- When ARE is 1 for the Security state of an interrupt, the field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by GICR_IGROUPR0.

When [GICD_CTLR.DS](#)==0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

Note

Accesses to GICD_IGROUPR0 when affinity routing is not enabled for a Security state access the same state as [GICR_IGROUPR0](#), and must update Redistributor state associated with the PE performing the accesses. Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICD_IGROUPR<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|-----------------|
| GIC Distributor | Dist_base | 0x0080 + (4 * n) | GICD_IGROUPR<n> |

Accesses on this interface are **RW**.

GICD_IGROUPR<n>E, Interrupt Group Registers (extended SPI range), n = 0 - 31

The GICD_IGROUPR<n>E characteristics are:

Purpose

Controls whether the corresponding SPI in the extended SPI range is in Group 0 or Group 1.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICD_IGROUPR<n>E are RES0.

When [GICD_TYPER.ESPI](#)==0, these registers are RES0.

When [GICD_TYPER.ESPI](#)==1:

- The number of implemented GICD_IGROUPR<n>E registers is ([GICD_TYPER.ESPI_range](#)+1). Registers are numbered from 0.
- When [GICD_CTLR.DS](#)==0, this register is Secure.

Attributes

GICD_IGROUPR<n>E is a 32-bit register.

Field descriptions

| | | | | | |
|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 |
| Group_status_bit31 | Group_status_bit30 | Group_status_bit29 | Group_status_bit28 | Group_status_bit27 | Group_status_bit26 |

Group_status_bit<x>, bit [x], for x = 31 to 0

Group status bit.

| Group_status_bit<x> | Meaning |
|---------------------|---|
| 0b0 | When GICD_CTLR.DS ==1, the corresponding interrupt is Group 0. When GICD_CTLR.DS ==0, the corresponding interrupt is Secure. |
| 0b1 | When GICD_CTLR.DS ==1, the corresponding interrupt is Group 1. When GICD_CTLR.DS ==0, the corresponding interrupt is Non-secure Group 1. |

If affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in [GICD_IGRPMODR<n>E](#) to form a 2-bit field that defines an interrupt group. The encoding of this field is described in [GICD_IGRPMODR<n>E](#).

If affinity routing is disabled for the Security state of an interrupt, the bit is RES0:

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IGROUPR<n>E number, n, is given by $n = (m-4096) \text{ DIV } 32$.
- The offset of the required GICD_IGROUPR<n>E is $(0 \times 1000 + (4 \times n))$.

- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.

Accessing GICD_IGROUPR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_IGROUPR<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)=0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_IGROUPR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|------------------|
| GIC Distributor | Dist_base | 0x1000 + (4 * n) | GICD_IGROUPR<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_IGRPMODR<n>, Interrupt Group Modifier Registers, n = 0 - 31

The GICD_IGRPMODR<n> characteristics are:

Purpose

When [GICD_CTLR.DS](#)==0, this register together with the [GICD_IGROUPR<n>](#) registers, controls whether the corresponding interrupt is in:

- Secure Group 0.
- Non-secure Group 1.
- Secure Group 1.

Configuration

When [GICD_CTLR.DS](#)==0, these registers are Secure.

The number of implemented [GICD_IGROUPR<n>](#) registers is ([GICD_TYPER.ITLinesNumber](#)+1). Registers are numbered from 0.

When [GICD_CTLR.ARE_S](#)==0 or [GICD_CTLR.DS](#)==1, the GICD_IGRPMODR<n> registers are RES0. An implementation can make these registers RAZ/WI in this case.

Attributes

GICD_IGRPMODR<n> is a 32-bit register.

Field descriptions

| | | | | | |
|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------|
| 31 | 30 | 29 | 28 | 27 | |
| Group_modifier_bit31 | Group_modifier_bit30 | Group_modifier_bit29 | Group_modifier_bit28 | Group_modifier_bit27 | Group... |

Group_modifier_bit<x>, bit [x], for x = 31 to 0

Group modifier bit. When affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in [GICD_IGROUPR<n>](#) to form a 2-bit field that defines an interrupt group:

| Group modifier bit | Group status bit | Definition | Short name |
|--------------------|------------------|---|------------|
| 0b0 | 0b0 | Secure Group 0 | G0S |
| 0b0 | 0b1 | Non-secure Group 1 | G1NS |
| 0b1 | 0b0 | Secure Group 1 | G1S |
| 0b1 | 0b1 | Reserved, treated as Non-secure Group 1 | - |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IGRPMODR<n> number, n, is given by $n = m \text{ DIV } 32$.
- The offset of the required GICD_IGRPMODR is $(0 \times 080 + (4 * n))$.
- The bit number of the required group modifier bit in this register is $m \text{ MOD } 32$.

See [GICD_IGROUPR<n>](#) for information about the GICD_IGRPMODR0 reset value.

Accessing GICD_IGRPMODR<n>

When affinity routing is enabled for Secure state, GICD_IGRPMODR0 is RES0 and equivalent functionality is proved by [GICR_IGRPMODR0](#).

When [GICD_CTLR](#).DS==0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICD_IGRPMODR<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|-------------------|------------------|
| GIC Distributor | Dist_base | 0x0D000 + (4 * n) | GICD_IGRPMODR<n> |

Accesses on this interface are **RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_IGRPMODR<n>E, Interrupt Group Modifier Registers (extended SPI range), n = 0 - 31

The GICD_IGRPMODR<n>E characteristics are:

Purpose

When [GICD_CTLR.DS](#)==0, this register together with the [GICD_IGROUPR<n>E](#) registers, controls whether the corresponding interrupt is in:

- Secure Group 0.
- Non-secure Group 1.
- When System register access is enabled, Secure Group 1.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICD_IGRPMODR<n>E are RES0.

GICD_IGRPMODR<n>E resets to 0x00000000.

When [GICD_TYPER.ESPI](#)==0, these registers are RES0.

When [GICD_TYPER.ESPI](#)==1:

- The number of implemented GICD_IGRPMODR<n>E registers is ([GICD_TYPER.ESPI_range](#)+1). Registers are numbered from 0.
- When [GICD_CTLR.DS](#)==0, this register is Secure.

Attributes

GICD_IGRPMODR<n>E is a 32-bit register.

Field descriptions

| 31 | 30 | 29 | 28 | 27 | |
|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|
| Group_modifier_bit31 | Group_modifier_bit30 | Group_modifier_bit29 | Group_modifier_bit28 | Group_modifier_bit27 | Group_modifier_bit26 |

Group_modifier_bit<x>, bit [x], for x = 31 to 0

Group modifier bit. In implementations where affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in [GICD_IGROUPR<n>E](#) to form a 2-bit field that defines an interrupt group:

| Group modifier bit | Group status bit | Definition | Short name |
|--------------------|------------------|---|------------|
| 0b0 | 0b0 | Secure Group 0 | G0S |
| 0b0 | 0b1 | Non-secure Group 1 | G1NS |
| 0b1 | 0b0 | Secure Group 1 | G1S |
| 0b1 | 0b1 | Reserved, treated as Non-secure Group 1 | - |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IGRPMODR<n>E number, n, is given by $n = (m-4096) \text{ DIV } 32$.
- The offset of the required GICD_IGRPMODR<n>E is $(0x3400 + (4*n))$.

- The bit number of the required group modifier bit in this register is (m-4096) MOD 32.

Accessing GICD_IGRPMODR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_IGRPMODR<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)=0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_IGRPMODR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------------|-------------------|
| GIC Distributor | Dist_base | 0x3400 + (4 * n) | GICD_IGRPMODR<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_IIDR, Distributor Implementer Identification Register

The GICD_IIDR characteristics are:

Purpose

Provides information about the implementer and revision of the Distributor.

Configuration

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states, this register is Common.

Attributes

GICD_IIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|------|----|----|----|---------|----|----|----|----------|----|----|----|-------------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ProductID | | | | | | | | RES0 | | | | Variant | | | | Revision | | | | Implementer | | | | | | | | | | | |

ProductID, bits [31:24]

Product Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [23:20]

Reserved, RES0.

Variant, bits [19:16]

Variant number. Typically, this field is used to distinguish product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

Revision number. Typically, this field is used to distinguish minor revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 code of the company that implemented the Distributor:

- Bits [11:8] are the JEP106 continuation code of the implementer. For an Arm implementation, this field is 0x4.
- Bit [7] is always 0.
- Bits [6:0] are the JEP106 identity code of the implementer. For an Arm implementation, bits [7:0] are therefore 0x3B.

Accessing GICD_IIDR

GICD_IIDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|--------------------|-----------|--------|-----------|
| GIC Distributor | Dist_base | 0x0008 | GICD_IIDR |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_INMIR<n>, Non-maskable Interrupt Registers, x = 0 to 31, n = 0 - 31

The GICD_INMIR<n> characteristics are:

Purpose

Holds whether the corresponding SPI has the non-maskable property.

Configuration

This register is present only when FEAT_GICv3_NMI is implemented. Otherwise, direct accesses to GICD_INMIR<n> are RES0.

When [GICR_TYPER](#).NMI is 0, this register is RES0.

The number of implemented GICD_INMIR<n> registers is ([GICD_TYPER](#).ITLinesNumber+1). Registers are numbered from 0.

Attributes

GICD_INMIR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | |
|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 |
| NMI31 | NMI30 | NMI29 | NMI28 | NMI27 | NMI26 | NMI25 | NMI24 | NMI23 | NMI22 | NMI21 | NMI20 | NMI19 | NMI18 | NMI17 | NMI16 | NMI15 | NMI14 |

NMI<x>, bit [x], for x = 31 to 0

Non-maskable property.

| NMI<x> | Meaning |
|--------|--|
| 0b0 | Interrupt does not have the non-maskable property. |
| 0b1 | Interrupt has the the non-maskable property. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_INMI<n> number, n, is given by $n = (m \text{ DIV } 32)$.
- The offset of the required GICD_INMI is $(0xF80 + (4*n))$.
- The bit number of the required in this register is $(m \text{ MOD } 32)$.

Accessing GICD_INMIR<n>

For SGIs and PPIs:

- The field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by GICR_INMIR0.

When affinity routing is not enabled for the Security state of an interrupt in GICD_IGROUPR<n>E, the corresponding bit is RES0.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_INMIR<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|---------------|
| GIC Distributor | Dist_base | 0x0F80 + (4 * n) | GICD_INMIR<n> |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_INMIR<n>E, Non-maskable Interrupt Registers for Extended SPIs, x = 0 to 31, n = 0 - 31

The GICD_INMIR<n>E characteristics are:

Purpose

Holds whether the corresponding SPI in the extended SPI range has the non-maskable property.

Configuration

This register is present only when FEAT_GICv3p1 is implemented and FEAT_GICv3_NMI is implemented. Otherwise, direct accesses to GICD_INMIR<n>E are RES0.

When [GICD_TYPER](#).ESPI is 0 or [GICD_TYPER](#).NMI is 0, these registers are RES0.

When [GICD_TYPER](#).ESPI is 1: the number of implemented GICD_INMIR<n>E registers is ([GICD_TYPER](#).ESPI_range+1). Registers are numbered from 0.

Attributes

GICD_INMIR<n>E is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | |
|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 |
| NMI31 | NMI30 | NMI29 | NMI28 | NMI27 | NMI26 | NMI25 | NMI24 | NMI23 | NMI22 | NMI21 | NMI20 | NMI19 | NMI18 | NMI17 | NMI16 | NMI15 | NMI14 |

NMI<x>, bit [x], for x = 31 to 0

Non-maskable property.

| NMI<x> | Meaning |
|--------|--|
| 0b0 | Interrupt does not have the non-maskable property. |
| 0b1 | Interrupt has the the non-maskable property. |

If affinity routing is disabled for the Security state of an interrupt, the bit is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_INMIR<n>E number, n, is given by $n = ((m-4096) \text{ DIV } 32)$.
- The offset of the required GICD_INMIR<n>E is $(0x3B00 + (4*n))$.
- The bit number in this register is $((m-4096) \text{ MOD } 32)$.

Accessing GICD_INMIR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_IGROUPR<n>E, the corresponding bit is RES0.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_INMIR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|----------------|
| GIC Distributor | Dist_base | 0x3B00 + (4 * n) | GICD_INMIR<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_IPRIORITYR<n>, Interrupt Priority Registers, n = 0 - 254

The GICD_IPRIORITYR<n> characteristics are:

Purpose

Holds the priority of the corresponding interrupt.

Configuration

These registers are available in all configurations of the GIC. When [GICD_CTLR.DS](#)==0, these registers are Common.

The number of implemented GICD_IPRIORITYR<n> registers is 8*([GICD_TYPER.ITLinesNumber](#)+1). Registers are numbered from 0.

GICD_IPRIORITYR0 to GICD_IPRIORITYR7 are Banked for each connected PE with [GICR_TYPER.Processor_Number](#) < 8.

Accessing GICD_IPRIORITYR0 to GICD_IPRIORITYR7 from a PE with [GICR_TYPER.Processor_Number](#) > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_IPRIORITYR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------|----|----|----|----|----|----|----|--------------------|----|----|----|----|----|----|----|--------------------|----|----|----|----|----|---|---|--------------------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Priority_offset_3B | | | | | | | | Priority_offset_2B | | | | | | | | Priority_offset_1B | | | | | | | | Priority_offset_0B | | | | | | | |

Priority_offset_3B, bits [31:24]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 3. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Priority_offset_2B, bits [23:16]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 2. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Priority_offset_1B, bits [15:8]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 1. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Priority_offset_0B, bits [7:0]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 0. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IPRIORITYR<n> number, n, is given by $n = m \text{ DIV } 4$.
- The offset of the required GICD_IPRIORITYR<n> register is $(0x400 + (4*n))$.
- The byte offset of the required Priority field in this register is $m \text{ MOD } 4$, where:
 - Byte offset 0 refers to register bits [7:0].
 - Byte offset 1 refers to register bits [15:8].
 - Byte offset 2 refers to register bits [23:16].
 - Byte offset 3 refers to register bits [31:24].

Accessing GICD_IPRIORITYR<n>

These registers are always used when affinity routing is not enabled. When affinity routing is enabled for the Security state of an interrupt:

- [GICR_IPRIORITYR<n>](#) is used instead of GICD_IPRIORITYR<n> where n = 0 to 7 (that is, for SGIs and PPIs).
- GICD_IPRIORITYR<n> is RAZ/WI where n = 0 to 7.

These registers are byte-accessible.

A register field corresponding to an unimplemented interrupt is RAZ/WI.

A GIC might implement fewer than eight priority bits, but must implement at least bits [7:4] of each field. In each field, unimplemented bits are RAZ/WI, see 'Interrupt prioritization' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

When [GICD_CTLR.DS](#)==0:

- A register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.
- A Non-secure access to a field that corresponds to a Non-secure Group 1 interrupt behaves as described in 'Software accesses of interrupt priority' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

It is IMPLEMENTATION DEFINED whether changing the value of a priority field changes the priority of an active interrupt.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICD_IPRIORITYR<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|--------------------|
| GIC Distributor | Dist_base | 0x0400 + (4 * n) | GICD_IPRIORITYR<n> |

Accesses on this interface are **RW**.

GICD_IPRIORITYR<n>E, Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.,
n = 0 - 255

GICD_IPRIORITYR<n>E, Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC., n = 0 - 255

The GICD_IPRIORITYR<n>E characteristics are:

Purpose

Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICD_IPRIORITYR<n>E are RES0.

When [GICD_TYPER](#).ESPI==0, these registers are RES0.

When [GICD_TYPER](#).ESPI==1, the number of implemented GICD_IPRIORITYR<n>E registers is (([GICD_TYPER](#).ESPI_range+1)*8). Registers are numbered from 0.

Attributes

GICD_IPRIORITYR<n>E is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------------------|----|----|----|----|----|----|----|------------------------------------|----|----|----|----|----|----|----|------------------------------------|----|----|----|----|----|---|---|------------------------------------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Priority_offset_3B | | | | | | | | Priority_offset_2B | | | | | | | | Priority_offset_1B | | | | | | | | Priority_offset_0B | | | | | | | |

Priority_offset_3B, bits [31:24]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 3. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_2B, bits [23:16]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 2. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_1B, bits [15:8]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 1. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

GICD_IPRIORITYR<n>E, Holds the priority of the corresponding interrupt for each extended SPI supported by the GIC.,
n = 0 - 255

Priority_offset_0B, bits [7:0]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 0. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_IPRIORITYR<n> number, n, is given by $n = (m-4096) \text{ DIV } 4$.
- The offset of the required GICD_IPRIORITYR<n>E register is $(0x2000 + (4*n))$.
- The byte offset of the required Priority field in this register is $m \text{ MOD } 4$, where:
 - Byte offset 0 refers to register bits [7:0].
 - Byte offset 1 refers to register bits [15:8].
 - Byte offset 2 refers to register bits [23:16].
 - Byte offset 3 refers to register bits [31:24].

Accessing GICD_IPRIORITYR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ISACTIVER<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)=0:

- A field that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.
- A Non-secure access to a field that corresponds to a Non-secure Group 1 interrupt behaves as described in Software accesses of interrupt priority.

Bits corresponding to unimplemented interrupts are RAZ/WI.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than once. The effect of the change must be visible in finite time.

GICD_IPRIORITYR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|---------------------|
| GIC Distributor | Dist_base | 0x2000 + (4 * n) | GICD_IPRIORITYR<n>E |

Accesses on this interface are **RW**.

GICD_IROUTER<n>, Interrupt Routing Registers, n = 32 - 1019

The GICD_IROUTER<n> characteristics are:

Purpose

When affinity routing is enabled, provides routing information for the SPI with INTID n.

Configuration

These registers are available in all configurations of the GIC. If the GIC implementation supports two Security states, these registers are Common.

The maximum value of n is given by (32*(GICD_TYPER.ITLinesNumber+1) - 1). [GICD_IROUTER<n>](#) registers where n=0 to 31 are reserved.

Attributes

GICD_IROUTER<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Aff3 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Aff0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Aff1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Aff2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Interrupt_Routing_Mode | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Interrupt_Routing_Mode, bit [31]

Interrupt Routing Mode. Defines how SPIs are routed in an affinity hierarchy:

| Interrupt_Routing_Mode | Meaning |
|------------------------|---|
| 0b0 | Interrupts routed to the PE specified by a.b.c.d. In this routing, a, b, c, and d are the values of fields Aff3, Aff2, Aff1, and Aff0 respectively. |
| 0b1 | Interrupts routed to any PE defined as a participating node. |

If GICD_IROUTER<n>.IRM == 0 and the affinity path does not correspond to an implemented PE, then if the corresponding interrupt becomes pending behavior is CONSTRAINED UNPREDICTABLE:

- The interrupt is not forwarded to any PE, direct reads return the written value

- The affinity path is treated as an UNKNOWN implemented PE, direct reads return the UNKNOWN implemented PE
- The affinity path is treated as an UNKNOWN implemented PE, direct reads return the written value

In implementations that do not require 1 of N distribution of SPIs, this bit might be RAZ/WI.

When this bit is set to 1, GICD_IROUTER<n>.{Aff3, Aff2, Aff1, Aff0} are UNKNOWN.

Note

An implementation might choose to make the Aff<n> fields RO when this field is 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [30:24]

Reserved, RES0.

Aff2, bits [23:16]

Affinity level 2.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Aff1, bits [15:8]

Affinity level 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Aff0, bits [7:0]

Affinity level 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

For an SPI with INTID m:

- The corresponding GICD_IROUTER<n> register number, n, is given by $n = m$.
- The offset of the GICD_IROUTER<n> register is $0x6000 + 8n$.

Accessing GICD_IROUTER<n>

These registers are used only when affinity routing is enabled. When affinity routing is not enabled:

- These registers are RES0. An implementation is permitted to make the register RAZ/WI in this case.
- The [GICD_ITARGETSR<n>](#) registers provide interrupt routing information.

Note

When affinity routing becomes enabled for a Security state (for example, following a reset or following a write to [GICD_CTLR](#)) the value of all writeable fields in this register is UNKNOWN for that Security state. When the group of an

interrupt changes so the ARE setting for the interrupt changes to 1, the value of this register is UNKNOWN for that interrupt.

If [GICD_CTLR.DS](#)==0, unless the [GICD_NSACR<n>](#) registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any GICD_IROUTER<n> registers that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

Note

For each interrupt, a GIC implementation might support fewer than 256 values for an affinity level. In this case, some bits of the corresponding affinity level field might be RO. Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICD_IROUTER<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|-----------------|
| GIC Distributor | Dist_base | 0x6000 + (8 * n) | GICD_IROUTER<n> |

Accesses on this interface are **RW**.

GICD_IROUTER<n>E, Interrupt Routing Registers (Extended SPI Range), n = 0 - 1023

The GICD_IROUTER<n>E characteristics are:

Purpose

When affinity routing is enabled, provides routing information for the corresponding SPI in the extended SPI range.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICD_IROUTER<n>E are RES0.

When [GICD_TYPER](#).ESPI==0, these registers are RES0.

When [GICD_TYPER](#).ESPI==1, the number of implemented GICD_IROUTER<n>E registers is ((([GICD_TYPER](#).ESPI_range+1)*32)-1). Registers are numbered from 0.

Attributes

GICD_IROUTER<n>E is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|------|----|----|----|------|----|----|----|--|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | Aff3 | | | | | | | | | | | |
| Interrupt_Routing_Mode | | | | RES0 | | | | | | | | Aff2 | | | | | | | | Aff1 | | | | | | | | Aff0 | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |

Bits [63:40]

Reserved, RES0.

Aff3, bits [39:32]

Affinity level 3.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Interrupt_Routing_Mode, bit [31]

Interrupt Routing Mode. Defines how SPIs are routed in an affinity hierarchy:

| Interrupt_Routing_Mode | Meaning |
|------------------------|---|
| 0b0 | Interrupts routed to the PE specified by a.b.c.d. In this routing, a, b, c, and d are the values of fields Aff3, Aff2, Aff1, and Aff0 respectively. |
| 0b1 | Interrupts routed to any PE defined as a participating node. |

If GICD_IROUTER<n>E.IRM == 0 and the affinity path does not correspond to an implemented PE, then if the corresponding interrupt becomes pending behavior is CONSTRAINED UNPREDICTABLE:

GICD_IROUTER<n>E, Interrupt Routing Registers (Extended SPI Range), n = 0 - 1023

- The interrupt is not forwarded to any PE, direct reads return the written value
- The affinity path is treated as an UNKNOWN implemented PE, direct reads return the UNKNOWN implemented PE
- The affinity path is treated as an UNKNOWN implemented PE, direct reads return the written value

In implementations that do not require 1 of N distribution of SPIs, this bit might be RAZ/WI.

When this bit is set to 1, GICD_IROUTER<n>E.{Aff3, Aff2, Aff1, Aff0} are UNKNOWN.

Note

An implementation might choose to make the Aff<n> fields RO when this field is 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [30:24]

Reserved, RES0.

Aff2, bits [23:16]

Affinity level 2.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Aff1, bits [15:8]

Affinity level 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Aff0, bits [7:0]

Affinity level 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

For an SPI with INTID m:

- The corresponding GICD_IROUTER<n>E register number, n, is given by $n = m$.
- The offset of the GICD_IROUTER<n>E register is $0x6000 + 8n$.

Accessing GICD_IROUTER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_IROUTER<n>E, the register is RES0.

When [GICD_CTLR.DS=0](#), a register that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_IROUTER<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|------------------|
| GIC Distributor | Dist_base | 0x8000 + (8 * n) | GICD_IROUTER<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ISACTIVER<n>, Interrupt Set-Active Registers, n = 0 - 31

The GICD_ISACTIVER<n> characteristics are:

Purpose

Activates the corresponding interrupt. These registers are used when saving and restoring GIC state.

Configuration

These registers are available in all GIC configurations. If [GICD_CTLR.DS](#)==0, these registers are Common.

The number of implemented [GICD_ISACTIVER<n>](#) registers is ([GICD_TYPER.ITLinesNumber](#)+1). Registers are numbered from 0.

GICD_ISACTIVER0 is Banked for each connected PE with [GICR_TYPER.Processor_Number](#) < 8.

Accessing GICD_ISACTIVER0 from a PE with [GICR_TYPER.Processor_Number](#) > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ISACTIVER<n> is a 32-bit register.

Field descriptions

| | | | | | | |
|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 |
| Set_active_bit31 | Set_active_bit30 | Set_active_bit29 | Set_active_bit28 | Set_active_bit27 | Set_active_bit26 | Set_active_bit25 |

Set_active_bit<x>, bit [x], for x = 31 to 0

Adds the active state to interrupt number 32n + x. Reads and writes have the following behavior:

| Set_active_bit<x> | Meaning |
|-------------------|---|
| 0b0 | If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is active, or is active and pending. If written, activates the corresponding interrupt, if the interrupt is not already active. If the interrupt is already active, the write has no effect. After a write of 1 to this bit, a subsequent read of this bit returns 1. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISACTIVER<n> number, n, is given by n = m DIV 32.
- The offset of the required GICD_ISACTIVER is (0x300 + (4*n)).
- The bit number of the required group modifier bit in this register is m MOD 32.

Accessing GICD_ISACTIVER<n>

When affinity routing is enabled for the Security state of an interrupt, bits corresponding to SGIs and PPIs are RAZ/WI, and equivalent functionality for SGIs and PPIs is provided by [GICR_ISACTIVER0](#).

Bits corresponding to unimplemented interrupts are RAZ/WI.

If [GICD_CTLR.DS](#) == 0, unless the [GICD_NSACR<n>](#) registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

The bit reads as one if the status of the interrupt is active or active and pending. [GICD_ISPENDR<n>](#) and [GICD_ICPENDR<n>](#) provide the pending status of the interrupt.

GICD_ISACTIVER<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|-------------------|
| GIC Distributor | Dist_base | 0x0300 + (4 * n) | GICD_ISACTIVER<n> |

This interface is accessible as follows:

- When [GICD_CTLR.DS](#) == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ISACTIVER<n>E, Interrupt Set-Active Registers (extended SPI range), n = 0 - 31

The GICD_ISACTIVER<n>E characteristics are:

Purpose

Adds the active state to the corresponding SPI in the extended SPI range.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICD_ISACTIVER<n>E are RES0.

When GICD_TYPER.ESPI==0, these registers are RES0.

When GICD_TYPER.ESPI==1, the number of implemented GICD_ISACTIVER<n>E registers is (GICD_TYPER.ESPI_range+1). Registers are numbered from 0.

Attributes

GICD_ISACTIVER<n>E is a 32-bit register.

Field descriptions

| | | | | | | |
|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 |
| Set_active_bit31 | Set_active_bit30 | Set_active_bit29 | Set_active_bit28 | Set_active_bit27 | Set_active_bit26 | Set_active_bit25 |

Set_active_bit<x>, bit [x], for x = 31 to 0

For the extended SPIs, adds the active state to interrupt number x. Reads and writes have the following behavior:

| Set_active_bit<x> | Meaning |
|-------------------|---|
| 0b0 | If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is active, or active and pending on this PE. If written, activates the corresponding interrupt, if the interrupt is not already active. If the interrupt is already active, the write has no effect. After a write of 1 to this bit, a subsequent read of this bit returns 1. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISACTIVER<n>E number, n, is given by $n = (m-4096) \text{ DIV } 32$.
- The offset of the required GICD_ISACTIVER<n>E is $(0x1A00 + (4*n))$.
- The bit number of the required group modifier bit in this register is $(m-4096) \text{ MOD } 32$.

Accessing GICD_ISACTIVER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ISACTIVER<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)=0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_ISACTIVER<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|--------------------|
| GIC Distributor | Dist_base | 0x1A00 + (4 * n) | GICD_ISACTIVER<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ISENABLER<n>, Interrupt Set-Enable Registers, n = 0 - 31

The GICD_ISENABLER<n> characteristics are:

Purpose

Enables forwarding of the corresponding interrupt to the CPU interfaces.

Configuration

These registers are available in all GIC configurations. If [GICD_CTLR.DS](#)==0, these registers are Common.

The number of implemented GICD_ISENABLER<n> registers is ([GICD_TYPER.ITLinesNumber](#)+1). Registers are numbered from 0.

GICD_ISENABLER0 is Banked for each connected PE with [GICR_TYPER.Processor_Number](#) < 8.

Accessing GICD_ISENABLER0 from a PE with [GICR_TYPER.Processor_Number](#) > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ISENABLER<n> is a 32-bit register.

Field descriptions

| | | | | | | |
|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 |
| Set_enable_bit31 | Set_enable_bit30 | Set_enable_bit29 | Set_enable_bit28 | Set_enable_bit27 | Set_enable_bit26 | Set_enable_bit25 |

Set_enable_bit<x>, bit [x], for x = 31 to 0

For SPIs and PPIs, controls the forwarding of interrupt number 32n + x to the CPU interfaces. Reads and writes have the following behavior:

| Set_enable_bit<x> | Meaning |
|-------------------|---|
| 0b0 | If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect. |
| 0b1 | If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 1. |

For SGIs, the behavior of this bit is IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISENABLER<n> number, n, is given by n = m DIV 32.
- The offset of the required GICD_ISENABLER is (0x100 + (4*n)).
- The bit number of the required group modifier bit in this register is m MOD 32.

At start-up, and after a reset, a PE can use this register to discover which peripheral INTIDs the GIC supports. If [GICD_CTLR.DS](#)=0 in a system that supports EL3, the PE must do this for the Secure view of the available interrupts, and Non-secure software running on the PE must do this discovery after the Secure software has configured interrupts as Group 0/Secure Group 1 and Non-secure Group 1.

Accessing GICD_ISENABLER<n>

For SGIs and PPIs:

- When ARE is 1 for the Security state of an interrupt, the field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case.
- Equivalent functionality is provided by GICR_ISENABLER0.

Bits corresponding to unimplemented interrupts are RAZ/WI.

When [GICD_CTLR.DS](#)=0, bits corresponding to Group 0 or Secure Group 1 interrupts are RAZ/WI to Non-secure accesses.

It is IMPLEMENTATION DEFINED whether implemented SGIs are permanently enabled, or can be enabled and disabled by writes to [GICD_ISENABLER<n>](#) and [GICD_ICENABLER<n>](#) where n=0.

For SPIs and PPIs, each bit controls the forwarding of the corresponding interrupt from the Distributor to the CPU interfaces.

GICD_ISENABLER<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|-------------------|
| GIC Distributor | Dist_base | 0x0100 + (4 * n) | GICD_ISENABLER<n> |

Accesses on this interface are **RW**.

GICD_ISENABLER<n>E, Interrupt Set-Enable Registers, n = 0 - 31

The GICD_ISENABLER<n>E characteristics are:

Purpose

Enables forwarding of the corresponding SPI in the extended SPI range to the CPU interfaces.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICD_ISENABLER<n>E are RES0.

When [GICD_TYPER.ESPI](#)==0, these registers are RES0.

When [GICD_TYPER.ESPI](#)==1, the number of implemented [GICD_ISENABLER<n>E](#) registers is ([GICD_TYPER.ESPI_range](#)+1). Registers are numbered from 0.

Attributes

GICD_ISENABLER<n>E is a 32-bit register.

Field descriptions

| | | | | | | |
|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 |
| Set_enable_bit31 | Set_enable_bit30 | Set_enable_bit29 | Set_enable_bit28 | Set_enable_bit27 | Set_enable_bit26 | Set_enable_bit25 |

Set_enable_bit<x>, bit [x], for x = 31 to 0

For the extended SPI range, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

| Set_enable_bit<x> | Meaning |
|-------------------|---|
| 0b0 | If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect. |
| 0b1 | If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 1. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISENABLER<n>E number, n, is given by $n = (m-4096) \text{ DIV } 32$.
- The offset of the required GICD_ISENABLER<n>E is $(0x1200 + (4*n))$.
- The bit number of the required group modifier bit in this register is $(m-4096) \text{ MOD } 32$.

Accessing GICD_ISENABLER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ISENABLER<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)=0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_ISENABLER<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|--------------------|
| GIC Distributor | Dist_base | 0x1200 + (4 * n) | GICD_ISENABLER<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ISPENDR<n>, Interrupt Set-Pending Registers, n = 0 - 31

The GICD_ISPENDR<n> characteristics are:

Purpose

Adds the pending state to the corresponding interrupt.

Configuration

These registers are available in all GIC configurations. If [GICD_CTLR.DS](#)=0, these registers are Common.

The number of implemented [GICD_ISPENDR<n>](#) registers is ([GICD_TYPER.ITLinesNumber](#)+1). Registers are numbered from 0.

GICD_ISPENDR0 is Banked for each connected PE with [GICR_TYPER.Processor_Number](#) < 8.

Accessing GICD_ISPENDR0 from a PE with [GICR_TYPER.Processor_Number](#) > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ISPENDR<n> is a 32-bit register.

Field descriptions

| | | | | | | |
|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | |
| Set_pending_bit31 | Set_pending_bit30 | Set_pending_bit29 | Set_pending_bit28 | Set_pending_bit27 | Set_pending_bit26 | Set_pending_bit25 |

Set_pending_bit<x>, bit [x], for x = 31 to 0

For SPIs and PPIs, adds the pending state to interrupt number 32n + x. Reads and writes have the following behavior:

| Set pending bit<x> | Meaning |
|--------------------|---|
| 0b0 | If read, indicates that the corresponding interrupt is not pending on any PE. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is pending, or active and pending. If written, changes the state of the corresponding interrupt from inactive to pending, or from active to active and pending. This has no effect in the following cases: <ul style="list-style-type: none"> • If the interrupt is an SGI. The pending state of an SGI can be set using GICD_SPENDSGIR<n>. • If the interrupt is not inactive and is not active. • If the interrupt is already pending because of a write to GICD_ISPENDR<n>. • If the interrupt is already pending because the corresponding interrupt signal is asserted. In this case, the interrupt remains pending if the interrupt signal is deasserted. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Accessing GICD_ISPENDR<n>

Set-pending bits for SGIs are read-only and ignore writes. The Set-pending bits for SGIs are provided as [GICD_SPENDSGIR<n>](#).

When affinity routing is enabled for the Security state of an interrupt:

- Bits corresponding to SGIs and PPIs are RAZ/WI, and equivalent functionality for SGIs and PPIs is provided by GICR_ISPENDR0.
- Bits corresponding to Group 0 and Group 1 Secure interrupts can only be set by Secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

If [GICD_CTLR.DS](#)=0, unless the [GICD_NSACR<n>](#) registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

GICD_ISPENDR<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|-----------------|
| GIC Distributor | Dist_base | 0x0200 + (4 * n) | GICD_ISPENDR<n> |

Accesses on this interface are **RW**.

GICD_ISPENDR<n>E, Interrupt Set-Pending Registers (extended SPI range), n = 0 - 31

The GICD_ISPENDR<n>E characteristics are:

Purpose

Adds the pending state to the corresponding SPI in the extended SPI range.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICD_ISPENDR<n>E are RES0.

When [GICD_TYPER](#).ESPI==0, these registers are RES0.

When [GICD_TYPER](#).ESPI==1, the number of implemented GICD_ISPENDR<n>E registers is ([GICD_TYPER](#).ESPI_range+1). Registers are numbered from 0.

Attributes

GICD_ISPENDR<n>E is a 32-bit register.

Field descriptions

| | | | | | | |
|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|--------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | Se |
| Set_pending_bit31 | Set_pending_bit30 | Set_pending_bit29 | Set_pending_bit28 | Set_pending_bit27 | Set_pending_bit26 | Se |

Set_pending_bit<x>, bit [x], for x = 31 to 0

For the extended SPIs, adds the pending state to interrupt number x. Reads and writes have the following behavior:

| Set_pending_bit<x> | Meaning |
|--------------------|---|
| 0b0 | If read, indicates that the corresponding interrupt is not pending. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is pending, or active and pending. If written, changes the state of the corresponding interrupt from inactive to pending, or from active to active and pending. This has no effect in the following cases: <ul style="list-style-type: none">If the interrupt is already pending because of a write to GICD_ISPENDR<n>E.If the interrupt is already pending because the corresponding interrupt signal is asserted. In this case, the interrupt remains pending if the interrupt signal is deasserted. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ISPENDR<n>E number, n, is given by $n = (m-4096) \text{ DIV } 32$.

- The offset of the required GICD_ISPENDR<n>E is $(0 \times 1600 + (4 * n))$.
- The bit number of the required group modifier bit in this register is $(m - 4096) \text{ MOD } 32$.

Accessing GICD_ISPENDR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICD_ISPENDR<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)=0, bits corresponding to Secure SPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICD_ISPENDR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|---------------------------|------------------|
| GIC Distributor | Dist_base | $0 \times 1600 + (4 * n)$ | GICD_ISPENDR<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_ITARGETSR<n>, Interrupt Processor Targets Registers, n = 0 - 254

The GICD_ITARGETSR<n> characteristics are:

Purpose

When affinity routing is not enabled, holds the list of target PEs for the interrupt. That is, it holds the list of CPU interfaces to which the Distributor forwards the interrupt if it is asserted and has sufficient priority.

Configuration

These registers are available in all configurations of the GIC. When [GICD_CTLR.DS](#)==0, these registers are Common.

The number of implemented GICD_ITARGETSR<n> registers is 8*([GICD_TYPER.ITLinesNumber](#)+1). Registers are numbered from 0.

GICD_ITARGETSR0 to GICD_ITARGETSR7 are Banked for each connected PE with [GICR_TYPER.Processor_Number](#) < 8.

Accessing GICD_ITARGETSR0 to GICD_ITARGETSR7 from a PE with [GICR_TYPER.Processor_Number](#) > 7 is CONSTRAINED UNPREDICTABLE:

- Register is RAZ/WI.
- An UNKNOWN banked copy of the register is accessed.

Attributes

GICD_ITARGETSR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------------------------|----|----|----|----|----|----|----|---------------------------------------|----|----|----|----|----|----|----|---------------------------------------|----|----|----|----|----|---|---|---------------------------------------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CPU_targets_offset_3B | | | | | | | | CPU_targets_offset_2B | | | | | | | | CPU_targets_offset_1B | | | | | | | | CPU_targets_offset_0B | | | | | | | |

PEs in the system number from 0, and each bit in a PE targets field refers to the corresponding PE. For example, a value of 0x3 means that the Pending interrupt is sent to PEs 0 and 1. For GICD_ITARGETSR0-GICD_ITARGETSR7, a read of any targets field returns the number of the PE performing the read.

CPU_targets_offset_3B, bits [31:24]

PE targets for an interrupt, at byte offset 3.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

CPU_targets_offset_2B, bits [23:16]

PE targets for an interrupt, at byte offset 2.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

CPU_targets_offset_1B, bits [15:8]

PE targets for an interrupt, at byte offset 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

CPU_targets_offset_0B, bits [7:0]

PE targets for an interrupt, at byte offset 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

The bits that are set to 1 in the PE targets field determine which PEs are targeted:

| Value of PE targets field | Interrupt targets |
|---------------------------|-------------------|
| 0bxxxxxx1 | CPU interface 0 |
| 0bxxxxxx1x | CPU interface 1 |
| 0bxxxxxx1xx | CPU interface 2 |
| 0bxxxx1xxx | CPU interface 3 |
| 0bxxx1xxxx | CPU interface 4 |
| 0bxx1xxxxx | CPU interface 5 |
| 0bx1xxxxxx | CPU interface 6 |
| 0b1xxxxxxx | CPU interface 7 |

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_ITARGETSR<n> number, n, is given by $n = m \text{ DIV } 4$.
- The offset of the required GICD_ITARGETSR<n> register is $(0x800 + (4*n))$.
- The byte offset of the required Priority field in this register is $m \text{ MOD } 4$, where:
 - Byte offset 0 refers to register bits [7:0].
 - Byte offset 1 refers to register bits [15:8].
 - Byte offset 2 refers to register bits [23:16].
 - Byte offset 3 refers to register bits [31:24].

Software can write to these registers at any time. Any change to a targets field value:

- Has no effect on any active interrupt. This means that removing a CPU interface from a targets list does not cancel an active state for interrupts on that CPU interface. There is no effect on interrupts that are active and pending until the active status is cleared, at which time it is treated as a pending interrupt.
- Has an effect on any pending interrupts. This means:
 - Enables the CPU interface to be chosen as a target for the pending interrupt using an IMPLEMENTATION DEFINED mechanism.
 - Removing a CPU interface from the target list of a pending interrupt removes the pending state of the interrupt on that CPU interface.

Accessing GICD_ITARGETSR<n>

These registers are used when affinity routing is not enabled. When affinity routing is enabled for the Security state of an interrupt, the target PEs for an interrupt are defined by [GICD_IROUTER<n>](#) and the associated byte in GICD_ITARGETSR<n> is RES0. An implementation is permitted to make the byte RAZ/WI in this case.

- These registers are byte-accessible.
- A register field corresponding to an unimplemented interrupt is RAZ/WI.
- A field bit corresponding to an unimplemented CPU interface is RAZ/WI.
- GICD_ITARGETSR0-GICD_ITARGETSR7 are read-only. Each field returns a value that corresponds only to the PE reading the register.
- It is IMPLEMENTATION DEFINED which, if any, SPIs are statically configured in hardware. The field for such an SPI is read-only, and returns a value that indicates the PE targets for the interrupt.
- If [GICD_CTLR.DS](#)==0, unless the [GICD_NSACR<n>](#) registers permit Non-secure software to control Group 0 and Secure Group 1 interrupts, any bits that correspond to Group 0 or Secure Group 1 interrupts are accessible only by Secure accesses and are RAZ/WI to Non-secure accesses.

In a single connected PE implementation, all interrupts target one PE, and these registers are RAZ/WI.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICD_ITARGETSR<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|-------------------|
| GIC Distributor | Dist_base | 0x0800 + (4 * n) | GICD_ITARGETSR<n> |

Accesses on this interface are **RW**.

GICD_NSACR<n>, Non-secure Access Control Registers, n = 0 - 63

The GICD_NSACR<n> characteristics are:

Purpose

Enables Secure software to permit Non-secure software on a particular PE to create and control Group 0 interrupts.

Configuration

The concept of selective enabling of Non-secure access to Group 0 and Secure Group 1 interrupts applies to SGIs and SPIs.

GICD_NSACR0 is a Banked register used for SGIs. A copy is provided for every PE that has a CPU interface and that supports this feature.

Attributes

GICD_NSACR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | |
|-------------|-------------|-------------|-------------|-------------|-------------|------------|------------|------------|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 |
| NS_access15 | NS_access14 | NS_access13 | NS_access12 | NS_access11 | NS_access10 | NS_access9 | NS_access8 | NS_access7 | | | | | | | | | |

NS_access<x>, bits [2x+1:2x], for x = 15 to 0

Controls Non-secure access of the interrupt with ID $16n + x$.

If the corresponding interrupt does not support configurable Non-secure access, the field is RAZ/WI.

Otherwise, the field is RW and determines the level of Non-secure control permitted if the interrupt is a Secure interrupt. If the interrupt is a Non-secure interrupt, this field is ignored.

The possible values of each 2-bit field are:

| NS_access<x> | Meaning |
|--------------|--|
| 0b00 | No Non-secure access is permitted to fields associated with the corresponding interrupt. |
| 0b01 | Non-secure read and write access is permitted to set-pending bits in GICD_ISPENDR<n> associated with the corresponding interrupt. A Non-secure write access to GICD_SETSPI_NSR is permitted to set the pending state of the corresponding interrupt. A Non-secure write access to GICD_SGIR is permitted to generate a Secure SGI for the corresponding interrupt. An implementation might also provide read access to clear-pending bits in GICD_ICPENDR<n> associated with the corresponding interrupt. |
| 0b10 | As 0b01, but adds Non-secure read and write access permission to fields associated with the corresponding interrupt in the GICD_ICPENDR<n> registers. A Non-secure write access to GICD_CLRSPI_NSR is permitted to clear the pending state of the corresponding interrupt. Also adds Non-secure read access permission to fields associated with the corresponding interrupt in the GICD_ISACTIVER<n> and GICD_ICACTIVER<n> registers. |
| 0b11 | For GICD_NSACR0 this encoding is reserved and treated as 10. For all other GICD_NSACR<n> registers this encoding is treated as 0b10, but adds Non-secure read and write access permission to GICD_ITARGETSR<n> and GICD_IROUTER<n> fields associated with the corresponding interrupt. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_NSACR<n> number, n, is given by $n = m \text{ DIV } 16$.
- The offset of the required GICD_NSACR<n> register is $(0xE00 + (4*n))$.

Note

Because each field in this register comprises two bits, GICD_NSACR0 controls access rights to SGI registers, GICD_NSACR1 controls access to PPI registers (and is always RAZ/WI), and all other GICD_NSACR<n> registers control access to SPI registers.

For compatibility with GICv2, writes to GICD_NSACR0 for a particular PE must be coordinated within the Distributor and must update [GICR_NSACR](#) for the Redistributor associated with that PE.

Accessing GICD_NSACR<n>

These registers are always used when affinity routing is not enabled. When affinity routing is enabled for the Secure state, GICD_NSACR0 is RES0 and [GICR_NSACR](#) provides equivalent functionality for SGIs.

These registers do not support PPIs, therefore GICD_NSACR1 is RAZ/WI.

GICD_NSACR<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|-------------------|---------------|
| GIC Distributor | Dist_base | $0xE00 + (4 * n)$ | GICD_NSACR<n> |

This interface is accessible as follows:

- When GICD_CTLR.DS == 1 accesses to this register are **RAZ/WI**.
- When GICD_CTLR.DS == 0 and an access is Secure accesses to this register are **RW**.
- When GICD_CTLR.DS == 0 and an access is Non-secure accesses to this register are **RAZ/WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRoot() accesses to this register are **RW**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRealm() accesses to this register are **RAZ/WI**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_NSACR<n>E, Non-secure Access Control Registers, n = 0 - 63

The GICD_NSACR<n>E characteristics are:

Purpose

Enables Secure software to permit Non-secure software on a particular PE to create and control Group 0 interrupts.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICD_NSACR<n>E are RES0.

When [GICD_TYPER](#).ESPI==0, these registers are RES0.

When [GICD_TYPER](#).ESPI==1, the number of implemented GICD_ICFGR<n>E registers is (([GICD_TYPER](#).ESPI_range+1)*2). Registers are numbered from 0.

Attributes

GICD_NSACR<n>E is a 32-bit register.

Field descriptions

| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 |
|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|----------------------------|----------------------------|----------------------------|----|----|----|----|----|----|----|----|----|
| NS_access15 | NS_access14 | NS_access13 | NS_access12 | NS_access11 | NS_access10 | NS_access9 | NS_access8 | NS_access7 | | | | | | | | | |

NS_access<x>, bits [2x+1:2x], for x = 15 to 0

Controls Non-secure access of the interrupt with ID 16n + x.

If the corresponding interrupt does not support configurable Non-secure access, the field is RAZ/WI.

Otherwise, the field is RW and determines the level of Non-secure control permitted if the interrupt is a Secure interrupt. If the interrupt is a Non-secure interrupt, this field is ignored.

The possible values of each 2-bit field are:

| NS_access<x> | Meaning |
|--------------|---|
| 0b00 | No Non-secure access is permitted to fields associated with the corresponding interrupt. |
| 0b01 | Non-secure read and write access is permitted to set-pending bits in GICD_ISPENDR<n>E associated with the corresponding interrupt. A Non-secure write access to GICD_SETSPI_NSR is permitted to set the pending state of the corresponding interrupt. |
| 0b10 | As 0b01, but adds Non-secure read and write access permission to fields associated with the corresponding interrupt in the GICD_ICPENDR<n>E registers. A Non-secure write access to GICD_CLRSPI_NSR is permitted to clear the pending state of the corresponding interrupt. Also adds Non-secure read access permission to fields associated with the corresponding interrupt in the GICD_ISACTIVER<n>E and GICD_ICACTIVER<n>E registers. |
| 0b11 | This encoding is treated as 0b10, but adds Non-secure read and write access permission to GICD_IROUTER<n>E fields associated with the corresponding interrupt. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_NSACR<n>E number, n, is given by $n = (m - 4096) \text{ DIV } 16$.
- The offset of the required GICD_NSACR<n>E register is $(0 \times 3600 + (4 * n))$.

Accessing GICD_NSACR<n>E

GICD_NSACR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|---------------------------|----------------|
| GIC Distributor | Dist_base | $0 \times 3600 + (4 * n)$ | GICD_NSACR<n>E |

This interface is accessible as follows:

- When GICD_CTLR.DS == 1 accesses to this register are **RAZ/WI**.
- When GICD_CTLR.DS == 0 and an access is Secure accesses to this register are **RW**.
- When GICD_CTLR.DS == 0 and an access is Non-secure accesses to this register are **RAZ/WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRoot() accesses to this register are **RW**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRealm() accesses to this register are **RAZ/WI**.

GICD_SETSPI_NSR, Set Non-secure SPI Pending Register

The GICD_SETSPI_NSR characteristics are:

Purpose

Adds the pending state to a valid SPI if permitted by the Security state of the access and the [GICD_NSACR<n>](#) value for that SPI.

A write to this register changes the state of an inactive SPI to pending, and the state of an active SPI to active and pending.

Configuration

If [GICD_TYPER](#).MBIS == 0, this register is reserved.

When [GICD_CTLR](#).DS == 1, this register provides functionality for all SPIs.

Attributes

GICD_SETSPI_NSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | |

Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

The INTID of the SPI.

The function of this register depends on whether the targeted SPI is configured to be an edge-triggered or level-sensitive interrupt:

- For an edge-triggered interrupt, a write to GICD_SETSPI_NSR or [GICD_SETSPI_SR](#) adds the pending state to the targeted interrupt. It will stop being pending on activation, or if the pending state is removed by a write to [GICD_CLRSPI_NSR](#), [GICD_CLRSPI_SR](#), or [GICD_ICPENDR<n>](#).
- For a level-sensitive interrupt, a write to GICD_SETSPI_NSR or [GICD_SETSPI_SR](#) adds the pending state to the targeted interrupt. It will remain pending until it is deasserted by a write to [GICD_CLRSPI_NSR](#) or [GICD_CLRSPI_SR](#). If the interrupt is activated between having the pending state added and being deactivated, then the interrupt will be active and pending.

Accessing GICD_SETSPI_NSR

Writes to this register have no effect if:

- The value written specifies a Secure SPI, the value is written by a Non-secure access, and the value of the corresponding [GICD_NSACR<n>](#) register is 0.
- The value written specifies an invalid SPI.
- The SPI is already pending.

16-bit accesses to bits [15:0] of this register must be supported.

Note

A Secure access to this register can set the pending state of any valid SPI.

GICD_SETSPI_NSR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|--------|-----------------|
| GIC Distributor | Dist_base | 0x0040 | GICD_SETSPI_NSR |

Accesses on this interface are **WO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_SETSPI_SR, Set Secure SPI Pending Register

The GICD_SETSPI_SR characteristics are:

Purpose

Adds the pending state to a valid SPI.

A write to this register changes the state of an inactive SPI to pending, and the state of an active SPI to active and pending.

Configuration

If [GICD_TYPER](#).MBIS == 0, this register is reserved.

When [GICD_CTLR](#).DS == 1, this register is WI.

Attributes

GICD_SETSPI_SR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | |

Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

The INTID of the SPI.

The function of this register depends on whether the targeted SPI is configured to be an edge-triggered or level-sensitive interrupt:

- For an edge-triggered interrupt, a write to [GICD_SETSPI_NSR](#) or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will stop being pending on activation, or if the pending state is removed by a write to [GICD_CLRSPI_NSR](#), [GICD_CLRSPI_SR](#), or [GICD_ICPENDR<n>](#).
- For a level-sensitive interrupt, a write to [GICD_SETSPI_NSR](#) or GICD_SETSPI_SR adds the pending state to the targeted interrupt. It will remain pending until it is deasserted by a write to [GICD_CLRSPI_NSR](#) or [GICD_CLRSPI_SR](#). If the interrupt is activated between having the pending state added and being deactivated, then the interrupt will be active and pending.

Accessing GICD_SETSPI_SR

Writes to this register have no effect if:

- The value is written by a Non-secure access.
- The value written specifies an invalid SPI.
- The SPI is already pending.

16-bit accesses to bits [15:0] of this register must be supported.

GICD_SETSPI_SR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|--------|----------------|
| GIC Distributor | Dist_base | 0x0050 | GICD_SETSPI_SR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 1 accesses to this register are **WI**.
- When GICD_CTLR.DS == 0 and an access is Secure accesses to this register are **WO**.
- When GICD_CTLR.DS == 0 and an access is Non-secure accesses to this register are **WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRoot() accesses to this register are **WO**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRealm() accesses to this register are **WI**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_SGIR, Software Generated Interrupt Register

The GICD_SGIR characteristics are:

Purpose

Controls the generation of SGIs.

Configuration

This register is available in all configurations of the GIC. If the GIC supports two Security states this register is Common.

Attributes

GICD_SGIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|------------------|----|----|----|---------------|----|----|----|----|----|-------|----|------|----|----|----|---|---|-------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | TargetListFilter | | | | CPUTargetList | | | | | | NSATT | | RES0 | | | | | | INTID | | | | | | | |

Bits [31:26]

Reserved, RES0.

TargetListFilter, bits [25:24]

Determines how the Distributor processes the requested SGI.

| TargetListFilter | Meaning |
|------------------|---|
| 0b00 | Forward the interrupt to the CPU interfaces specified by GICD_SGIR.CPUTargetList. |
| 0b01 | Forward the interrupt to all CPU interfaces except that of the PE that requested the interrupt. |
| 0b10 | Forward the interrupt only to the CPU interface of the PE that requested the interrupt. |
| 0b11 | Reserved. |

CPUTargetList, bits [23:16]

When GICD_SGIR.TargetListFilter is 0b00, this field defines the CPU interfaces to which the Distributor must forward the interrupt.

Each bit of the field refers to the corresponding CPU interface. For example, CPUTargetList[0] corresponds to interface 0. Setting a bit to 1 indicates that the interrupt must be forwarded to the corresponding interface.

If this field is 0b00000000 when GICD_SGIR.TargetListFilter is 0b00, the Distributor does not forward the interrupt to any CPU interface.

NSATT, bit [15]

Specifies the required group of the SGI.

| NSATT | Meaning |
|-------|---|
| 0b0 | Forward the SGI specified in the INTID field to a specified CPU interface only if the SGI is configured as Group 0 on that interface. |
| 0b1 | Forward the SGI specified in the INTID field to a specified CPU interface only if the SGI is configured as Group 1 on that interface. |

This field is writable only by a Secure access. Non-secure accesses can also generate Group 0 interrupts, if allowed to do so by GICD_NSACR0. Otherwise, Non-secure writes to GICD_SGIR generate an SGI only if the specified SGI is programmed as Group 1, regardless of the value of bit [15] of the write.

Bits [14:4]

Reserved, RES0.

INTID, bits [3:0]

The INTID of the SGI to forward to the specified CPU interfaces.

Accessing GICD_SGIR

This register is used only when affinity routing is not enabled. When affinity routing is enabled, this register is RES0.

It is IMPLEMENTATION DEFINED whether this register has any effect when the forwarding of interrupts by the Distributor is disabled by [GICD_CTLR](#).

GICD_SGIR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|--------|-----------|
| GIC Distributor | Dist_base | 0x0F00 | GICD_SGIR |

Accesses on this interface are **WO**.

GICD_SPENDSGIR<n>, SGI Set-Pending Registers, n = 0 - 3

The GICD_SPENDSGIR<n> characteristics are:

Purpose

Adds the pending state to an SGI.

A write to this register changes the state of an inactive SGI to pending, and the state of an active SGI to active and pending.

Configuration

Four SGI set-pending registers are implemented. Each register contains eight set-pending bits for each of four SGIs, for a total of 16 possible SGIs.

In multiprocessor implementations, each PE has a copy of these registers.

Attributes

GICD_SPENDSGIR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------|----|----|----|----|----|----|----|-----------------------|----|----|----|----|----|----|----|-----------------------|----|----|----|----|----|---|---|-----------------------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SGI_set_pending_bits3 | | | | | | | | SGI_set_pending_bits2 | | | | | | | | SGI_set_pending_bits1 | | | | | | | | SGI_set_pending_bits0 | | | | | | | |

SGI_set_pending_bits<x>, bits [8x+7:8x], for x = 3 to 0

Adds the pending state to SGI number 4n + x for the PE corresponding to the bit number written to.

Reads and writes have the following behavior:

| SGI_set_pending_bits<x> | Meaning |
|-------------------------|---|
| 0x00 | If read, indicates that the SGI from the corresponding PE is not pending and is not active and pending. If written, has no effect. |
| 0x01 | If read, indicates that the SGI from the corresponding PE is pending or is active and pending. If written, adds the pending state to the SGI for the corresponding PE. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For SGI ID m, generated by processing element C writing to the corresponding [GICD_SGIR](#) field, where DIV and MOD are the integer division and modulo operations:

- The corresponding GICD_SPENDSGIR<n> number is given by n = m DIV 4.
- The offset of the required register is (0xF20 + (4n)).
- The offset of the required field within the register GICD_SPENDSGIR<n> is given by m MOD 4.
- The required bit in the 8-bit SGI set-pending field m is bit C.

Accessing GICD_SPENDSGIR<n>

These registers are used only when affinity routing is not enabled. When affinity routing is enabled for the Security state of an interrupt then the bit associated with SGI in that Security state is RES0. An implementation is permitted to make the register RAZ/WI in this case.

A register bit that corresponds to an unimplemented SGI is RAZ/WI.

These registers are byte-accessible.

If the GIC implementation supports two Security states:

- A register bit that corresponds to a Group 0 interrupt is RAZ/WI to Non-secure accesses.
- Register bits corresponding to unimplemented PEs are RAZ/WI.

GICD_SPENDSGIR<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|------------------|-------------------|
| GIC Distributor | Dist_base | 0x0F20 + (4 * n) | GICD_SPENDSGIR<n> |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_STATUSR, Error Reporting Status Register

The GICD_STATUSR characteristics are:

Purpose

Provides software with a mechanism to detect:

- Accesses to reserved locations.
- Writes to read-only locations.
- Reads of write-only locations.

Configuration

If the GIC implementation supports two Security states this register is Banked to provide Secure and Non-secure copies.

Attributes

GICD_STATUSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|------|----|-----|----|-----|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | WROD | | RWOD | | WRD | | RRD | | | | | | | | | |

Bits [31:4]

Reserved, RES0.

WROD, bit [3]

Write to an RO location.

| WROD | Meaning |
|------|--|
| 0b0 | Normal operation. |
| 0b1 | A write to an RO location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

RWOD, bit [2]

Read of a WO location.

| RWOD | Meaning |
|------|--|
| 0b0 | Normal operation. |
| 0b1 | A read of a WO location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

WRD, bit [1]

Write to a reserved location.

| WRD | Meaning |
|-----|---|
| 0b0 | Normal operation. |
| 0b1 | A write to a reserved location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

RRD, bit [0]

Read of a reserved location.

| RRD | Meaning |
|-----|--|
| 0b0 | Normal operation. |
| 0b1 | A read of a reserved location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

Accessing GICD_STATUSR

This is an optional register. If the register is not implemented, the location is RAZ/WI.

GICD_STATUSR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|--------|------------------|
| GIC Distributor | Dist_base | 0x0010 | GICD_STATUSR (S) |

This interface is accessible as follows:

- When an access is Secure accesses to this register are **RW**.
- When FEAT_RME is implemented and IsAccessRoot() accesses to this register are **RW**.

| Component | Frame | Offset | Instance |
|-----------------|-----------|--------|-------------------|
| GIC Distributor | Dist_base | 0x0010 | GICD_STATUSR (NS) |

This interface is accessible as follows:

- When an access is Non-secure accesses to this register are **RW**.
- When FEAT_RME is implemented and IsAccessRealm() accesses to this register are **RAZ/WI**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_TYPER, Interrupt Controller Type Register

The GICD_TYPER characteristics are:

Purpose

Provides information about what features the GIC implementation supports. It indicates:

- Whether the GIC implementation supports two Security states.
- The maximum number of INTIDs that the GIC implementation supports.
- The number of PEs that can be used as interrupt targets.

Configuration

This register is available in all configurations of the GIC. When [GICD_CTLR.DS](#)==0, this register is Common.

Attributes

GICD_TYPER is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|-----|------|-----|--------|------|------|------|----------|--------------|-----|------|-----------|---------------|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ESPI_range | RSS | No1N | A3V | IDbits | DVIS | LPIS | MBIS | num_LPis | SecurityExtn | NMI | ESPI | CPUNumber | ITLinesNumber | | | | | | | | | | | | | | | | | | |

ESPI_range, bits [31:27]

When GICD_TYPER.ESPI == 1:

Indicates the maximum INTID in the Extended SPI range.

Maximum Extended SPI INTID is $(32 * (\text{ESPI_range} + 1) + 4095)$.

The ESPI_range field only indicates the maximum number of SPIs that the GIC implementation might support. This value determines the number of instances of the following interrupt registers:

- [GICD_IGROUPR<n>E](#).
- [GICD_ISENBALER<n>E](#).
- [GICD_ICENABLER<n>E](#).
- [GICD_ISPENDR<n>E](#).
- [GICD_ICPENDR<n>E](#).
- [GICD_ISACTIVER<n>E](#).
- [GICD_ICACTIVER<n>E](#).
- [GICD_IPRIORITYR<n>E](#).
- [GICD_ICFGR<n>E](#).
- [GICD_IROUTER<n>E](#).
- [GICD_IGRPMODR<n>E](#).

The GIC architecture does not require a GIC implementation to support a continuous range of SPI interrupt IDs. Software must check which SPI INTIDs are supported, up to the maximum value indicated by GICD_TYPER.ESPI_range.

Otherwise:

Reserved, RES0.

RSS, bit [26]

Range Selector Support.

| RSS | Meaning |
|------------|---|
| 0b0 | The IRI supports targeted SGIs with affinity level 0 values of 0 - 15. |
| 0b1 | The IRI supports targeted SGIs with affinity level 0 values of 0 - 255. |

No1N, bit [25]

Indicates whether 1 of N SPI interrupts are supported.

| No1N | Meaning |
|-------------|--|
| 0b0 | 1 of N SPI interrupts are supported. |
| 0b1 | 1 of N SPI interrupts are not supported. |

A3V, bit [24]

Affinity 3 valid. Indicates whether the Distributor supports nonzero values of Affinity level 3.

| A3V | Meaning |
|------------|--|
| 0b0 | The Distributor only supports zero values of Affinity level 3. |
| 0b1 | The Distributor supports nonzero values of Affinity level 3. |

IDbits, bits [23:19]

The number of interrupt identifier bits supported, minus one.

DVIS, bit [18]

When FEAT_GICv4 is implemented:

Indicates whether the implementation supports Direct Virtual LPI injection.

| DVIS | Meaning |
|-------------|---|
| 0b0 | The implementation does not support Direct Virtual LPI injection. |
| 0b1 | The implementation supports Direct Virtual LPI injection. |

Otherwise:

Reserved, RES0.

LPIS, bit [17]

Indicates whether the implementation supports LPIS.

| LPIS | Meaning |
|-------------|---|
| 0b0 | The implementation does not support LPIS. |
| 0b1 | The implementation supports LPIS. |

MBIS, bit [16]

Indicates whether the implementation supports message-based interrupts by writing to Distributor registers.

| MBIS | Meaning |
|-------------|---|
| 0b0 | The implementation does not support message-based interrupts by writing to Distributor registers. The GICD_CLRSPI_NSR , GICD_SETSPI_NSR , GICD_CLRSPI_SR , and GICD_SETSPI_SR registers are reserved. |
| 0b1 | The implementation supports message-based interrupts by writing to the GICD_CLRSPI_NSR , GICD_SETSPI_NSR , GICD_CLRSPI_SR , or GICD_SETSPI_SR registers. |

num_LPIs, bits [15:11]

Number of supported LPis.

- 0b00000 Number of LPis as indicated by GICD_TYPER.IDbits.
- All other values Number of LPis supported is $2^{(\text{num_LPis}+1)}$.
 - Available LPI INTIDs are $8192..(8192 + 2^{(\text{num_LPis}+1)} - 1)$.
 - This field cannot indicate a maximum LPI INTID greater than that indicated by GICD_TYPER.IDbits.

When the supported INTID width is less than 14 bits, this field is RES0 and no LPis are supported.

SecurityExtn, bit [10]

Indicates whether the GIC implementation supports two Security states:

When [GICD_CTLR.DS](#) == 1, this field is RAZ.

| SecurityExtn | Meaning |
|--------------|---|
| 0b0 | The GIC implementation supports only a single Security state. |
| 0b1 | The GIC implementation supports two Security states. |

NMI, bit [9]

Non-maskable Interrupts.

| NMI | Meaning |
|-----|--|
| 0b0 | Non-maskable interrupt property not supported. |
| 0b1 | Non-maskable interrupt property is supported. |

ESPI, bit [8]

Extended SPI.

| ESPI | Meaning |
|------|-------------------------------------|
| 0b0 | Extended SPI range not implemented. |
| 0b1 | Extended SPI range implemented. |

CPUNumber, bits [7:5]

Reports the number of PEs that can be used when affinity routing is not enabled, minus 1.

These PEs must be numbered contiguously from zero, but the relationship between this number and the affinity hierarchy from MPIDR is IMPLEMENTATION DEFINED. If the implementation does not support ARE being zero, this field is 000.

ITLinesNumber, bits [4:0]

For the INTID range 32 to 1019, indicates the maximum SPI supported.

If the value of this field is N, the maximum SPI INTID is $32(N+1)$ minus 1. For example, 00011 specifies that the maximum SPI INTID is 127.

Regardless of the range of INTIDs defined by this field, interrupt IDs 1020-1023 are reserved for special purposes.

A value of 0 indicates no SPIs are support.

The ITLinesNumber field only indicates the maximum number of SPIs that the GIC implementation might support. This value determines the number of instances of the following interrupt registers:

- [GICD_IGROUPR<n>](#).
- [GICD_ISENBALER<n>](#).
- [GICD_ICENABLER<n>](#).
- [GICD_ISPENDR<n>](#).
- [GICD_ICPENDR<n>](#).
- [GICD_ISACTIVER<n>](#).
- [GICD_ICACTIVER<n>](#).
- [GICD_IPRIORITYR<n>](#).
- [GICD_ITARGETSR<n>](#).
- [GICD_ICFGR<n>](#).
- [GICD_IROUTER<n>](#).
- [GICD_IGRPMODR<n>](#).

The GIC architecture does not require a GIC implementation to support a continuous range of SPI interrupt IDs. Software must check which SPI INTIDs are supported, up to the maximum value indicated by GICD_TYPER.ITLinesNumber.

Accessing GICD_TYPER

GICD_TYPER can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|--------|------------|
| GIC Distributor | Dist_base | 0x0004 | GICD_TYPER |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICD_TYPER2, Interrupt Controller Type Register 2

The GICD_TYPER2 characteristics are:

Purpose

Provides information about which features the GIC implementation supports.

Configuration

This register is present only when FEAT_GICv4p1 is implemented. Otherwise, direct accesses to GICD_TYPER2 are RES0.

When [GICD_CTLR.DS](#) == 0, this register is Common.

Attributes

GICD_TYPER2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|-----------|---|-----|------|---|-----|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | nASSGICap | | VIL | RES0 | | VID | | | |

Bits [31:9]

Reserved, RES0.

nASSGICap, bit [8]

Indicates whether SGIs can be configured to not have an active state.

| nASSGICap | Meaning |
|-----------|--|
| 0b0 | SGIs have an active state. |
| 0b1 | SGIs can be globally configured not to have an active state. |

This bit is RES0 on implementations that support two Security states.

VIL, bit [7]

Indicates whether 16 bits of vPEID are implemented.

| VIL | Meaning |
|-----|---|
| 0b0 | GIC supports 16-bit vPEID. |
| 0b1 | GIC supports GICD_TYPER2.VID + 1 bits of vPEID. |

Bits [6:5]

Reserved, RES0.

VID, bits [4:0]

When GICD_TYPER2.VIL == 1, the number of bits is equal to the bits of vPEID minus one.

When GICD_TYPER2.VIL == 0, this field is RES0.

Accessing GICD_TYPER2

GICD_TYPER2 can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|-----------|--------|-------------|
| GIC Distributor | Dist_base | 0x000C | GICD_TYPER2 |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_APR<n>, Active Priorities Registers, n = 0 - 3

The GICH_APR<n> characteristics are:

Purpose

These registers track which preemption levels are active in the virtual CPU interface, and indicate the current active priority. Corresponding bits are set to 1 in this register when an interrupt is acknowledged, based on [GICH_LR<n>](#). Priority, and the least significant bit set is cleared on EOI.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_APR<n> are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

The number of registers required depends on how many bits are implemented in [GICH_LR<n>](#). Priority:

- When 5 priority bits are implemented, 1 register is required (GICH_APR0).
- When 6 priority bits are implemented, 2 registers are required (GICH_APR0, GICH_APR1).
- When 7 priority bits are implemented, 4 registers are required (GICH_APR0, GICH_APR1, GICH_APR2, GICH_APR3).

Unimplemented registers are RAZ/WI.

Attributes

GICH_APR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

P<x>, bit [x], for x = 31 to 0

Active priorities. Possible values of each bit are:

| P<x> | Meaning |
|------|---|
| 0b0 | There is no interrupt active at the priority corresponding to that bit. |
| 0b1 | There is an interrupt active at the priority corresponding to that bit. |

The correspondence between priorities and bits depends on the number of bits of priority that are implemented.

If 5 bits of priority are implemented (bits [7:3] of priority), then there are 32 priority groups, and the active state of these priorities are held in GICH_APR0 in the bits corresponding to Priority[7:3].

If 6 bits of priority are implemented (bits [7:2] of priority), then there are 64 priority groups, and:

- The active state of priorities 0 - 124 are held in GICH_APR0 in the bits corresponding to 0:Priority[6:2].
- The active state of priorities 128 - 252 are held in GICH_APR1 in the bits corresponding to 1:Priority[6:2].

If 7 bits of priority are implemented (bits [7:1] of priority), then there are 128 priority groups, and:

- The active state of priorities 0 - 62 are held in GICH_APR0 in the bits corresponding to 00:Priority[5:1].
- The active state of priorities 64 - 126 are held in GICH_APR1 in the bits corresponding to 01:Priority[5:1].
- The active state of priorities 128 - 190 are held in GICH_APR2 in the bits corresponding to 10:Priority[5:1].
- The active state of priorities 192 - 254 are held in GICH_APR3 in the bits corresponding to 11:Priority[5:1].

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing GICH_APR<n>

These registers are used only when System register access is not enabled. When System register access is enabled the following registers provide equivalent functionality:

- In AArch64:
 - For Group 0, [ICH_AP0R<n>_EL2](#).
 - For Group 1, [ICH_AP1R<n>_EL2](#).
- In AArch32:
 - For Group 0, [ICH_AP0R<n>](#).
 - For Group 1, [ICH_AP1R<n>](#).

GICH_APR<n> can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------------------|--------------------|-------------|
| GIC Virtual interface control | $0x00F0 + (4 * n)$ | GICH_APR<n> |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_EISR, End Interrupt Status Register

The GICH_EISR characteristics are:

Purpose

Indicates which List registers have outstanding EOI maintenance interrupts.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_EISR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICH_EISR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----------|----------|----------|----------|----------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | Status15 | Status14 | Status13 | Status12 | Status11 | Status10 | Status9 | Status8 | Status7 | Status6 | Status5 | Status4 | Status3 | Status2 | Status1 | Status0 |

Bits [31:16]

Reserved, RES0.

Status<n>, bit [n], for n = 15 to 0

EOI maintenance interrupt status for List register <n>:

| Status<n> | Meaning |
|-----------|--|
| 0b0 | GICH_LR<n> does not have an EOI maintenance interrupt. |
| 0b1 | GICH_LR<n> has an EOI maintenance interrupt that has not been handled. |

For any [GICH_LR<n>](#) register, the corresponding status bit is set to 1 if all of the following are true:

- [GICH_LR<n>](#).State is 0b00.
- [GICH_LR<n>](#).HW == 0.
- [GICH_LR<n>](#).EOI == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GICH_EISR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICH_EISR](#) provides equivalent functionality.
- For AArch64 implementations, [ICH_EISR_EL2](#) provides equivalent functionality.

Bits corresponding to unimplemented List registers are RAZ.

GICH_EISR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------------------|--------|-----------|
| GIC Virtual interface control | 0x0020 | GICH_EISR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_ELRSR, Empty List Register Status Register

The GICH_ELRSR characteristics are:

Purpose

Indicates which List registers contain valid interrupts.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_ELRSR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICH_ELRSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----------|----------|----------|----------|----------|---------|---------|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 |
| RES0 | | | | | | | | | | | | | | | | Status15 | Status14 | Status13 | Status12 | Status11 | Status10 | Status9 | Status8 | Status7 |

Bits [31:16]

Reserved, RES0.

Status<n>, bit [n], for n = 15 to 0

Status bit for List register <n>:

| Status<n> | Meaning |
|-----------|---|
| 0b0 | GICH_LR<n> , if implemented, contains a valid interrupt. Using this List register can result in overwriting a valid interrupt. |
| 0b1 | GICH_LR<n> does not contain a valid interrupt. The List register is empty and can be used without overwriting a valid interrupt or losing an EOI maintenance interrupt. |

For any [GICH_LR<n>](#) register, the corresponding status bit is set to 1 if [GICH_LR<n>](#).State is 0b00 and either:

- [GICH_LR<n>](#).HW == 1.
- [GICH_LR<n>](#).EOI == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 1.

Accessing GICH_ELRSR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICH_ELRSR](#) provides equivalent functionality.
- For AArch64 implementations, [ICH_ELRSR_EL2](#) provides equivalent functionality.

Bits corresponding to unimplemented List registers are RES0.

GICH_ELRSR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------------------|--------|------------|
| GIC Virtual interface control | 0x0030 | GICH_ELRSR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_HCR, Hypervisor Control Register

The GICH_HCR characteristics are:

Purpose

Controls the virtual CPU interface.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_HCR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICH_HCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | |
|--|------|---|---|----------|----------|----------|----------|------|---------|-----|----|
| 3130292827262524232221201918171615141312111098 | | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
| EOICount | RES0 | | | VGrp1DIE | VGrp1EIE | VGrp0DIE | VGrp0EIE | NPIE | LRENPIE | UIE | En |

EOICount, bits [31:27]

Counts the number of EOIs received that do not have a corresponding entry in the List registers. The virtual CPU interface increments this field automatically when a matching EOI is received. EOIs that do not clear a bit in [GICH_APR<n>](#) do not cause an increment. If an EOI occurs when the value of this field is 31, then the field wraps to 0.

The maintenance interrupt is asserted whenever this field is nonzero and GICH_HCR.LREN1E == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [26:8]

Reserved, RES0.

VGrp1DIE, bit [7]

VM Group 1 Disabled Interrupt Enable.

Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected virtual machine is disabled:

| VGrp1DIE | Meaning |
|----------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when GICV_CTLR.EnableGrp1 == 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VGrp1EIE, bit [6]

VM Group 1 Enabled Interrupt Enable.

Enables the signaling of a maintenance interrupt while signaling of Group 1 interrupts from the virtual CPU interface to the connected virtual machine is enabled:

| VGrp1EIE | Meaning |
|-----------------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when GICV_CTLR.EnableGrp1 == 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VGrp0DIE, bit [5]

VM Group 0 Disabled Interrupt Enable.

Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected virtual machine is disabled:

| VGrp0DIE | Meaning |
|-----------------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when GICV_CTLR.EnableGrp0 == 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VGrp0EIE, bit [4]

VM Group 0 Enabled Interrupt Enable.

Enables the signaling of a maintenance interrupt while signaling of Group 0 interrupts from the virtual CPU interface to the connected virtual machine is enabled:

| VGrp0EIE | Meaning |
|-----------------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled when GICV_CTLR.EnableGrp0 == 1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NPIE, bit [3]

No Pending Interrupt Enable.

Enables the signaling of a maintenance interrupt while no pending interrupts are present in the List registers:

| NPIE | Meaning |
|-------------|---|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled while the List registers contain no interrupts in the pending state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

LRENPIE, bit [2]

List Register Entry Not Present Interrupt Enable.

Enables the signaling of a maintenance interrupt while the virtual CPU interface does not have a corresponding valid List register for an EOI request:

| LRENPIE | Meaning |
|----------------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | Maintenance interrupt signaled while GICH_HCR.EOICount is not 0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

UIE, bit [1]

Underflow Interrupt Enable.

Enables the signaling of a maintenance interrupt when the List registers are either empty or hold only one valid entry.

| UIE | Meaning |
|------------|--|
| 0b0 | Maintenance interrupt disabled. |
| 0b1 | A maintenance interrupt is signaled if zero or one of the List register entries are marked as a valid interrupt. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

En, bit [0]

Enable.

Global enable bit for the virtual CPU interface.

| En | Meaning |
|-----------|--|
| 0b0 | Virtual CPU interface operation is disabled. |
| 0b1 | Virtual CPU interface operation is enabled. |

When this field is 0:

- The virtual CPU interface does not signal any maintenance interrupts.
- The virtual CPU interface does not signal any virtual interrupts.
- A read of [GICV_IAR](#) or [GICV_AIAR](#) returns a spurious interrupt ID.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The VGrp1DIE, VGrp1EIE, VGrp0DIE, and VGrp0EIE fields permit the hypervisor to track the virtual CPU interfaces that are enabled. The hypervisor can then route interrupts that have multiple targets correctly and efficiently, without having to read the virtual CPU interface status.

See 'Maintenance interrupts' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069) and [GICH_MISR](#) for more information.

Accessing GICH_HCR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICH_HCR](#) provides equivalent functionality.
- For AArch64 implementations, [ICH_HCR_EL2](#) provides equivalent functionality.

GICH_HCR.En must be set to 1 for any virtual or maintenance interrupt to be asserted.

GICH_HCR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------------------|--------|----------|
| GIC Virtual interface control | 0x0000 | GICH_HCR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_LR<n>, List Registers, n = 0 - 15

The GICH_LR<n> characteristics are:

Purpose

These registers provide context information for the virtual CPU interface.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_LR<n> are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

A maximum of 16 List registers can be provided. [GICH_VTR.ListRegs](#) defines the number implemented. Unimplemented List registers are RAZ/WI.

Attributes

GICH_LR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|-------|----------|----|----|----|----|------|----|----|--------|----|----|----|----|----|----|----|----|----|--------|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| HWGroup | | State | Priority | | | | | RES0 | | | pINTID | | | | | | | | | | vINTID | | | | | | | | | | |

HW, bit [31]

Indicates whether this virtual interrupt is a hardware interrupt, meaning that it corresponds to a physical interrupt. Deactivation of the virtual interrupt also causes the deactivation of the physical interrupt corresponding to the INTID:

| HW | Meaning |
|-----|--|
| 0b0 | This interrupt is triggered entirely in software. No notification is sent to the Distributor when the virtual interrupt is deactivated. |
| 0b1 | A hardware interrupt. A deactivate interrupt request is sent to the Distributor when the virtual interrupt is deactivated, using GICH_LR<n>.pINTID to indicate the physical interrupt identifier. If GICV_CTLR.EOImode == 0, this request corresponds to a write to GICV_EOIR or GICV_AEOIR , otherwise it corresponds to a write to GICV_DIR . |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Group, bit [30]

Indicates whether the interrupt is Group 0 or Group 1:

| Group | Meaning |
|-------|---|
| 0b0 | Group 0 virtual interrupt. GICV_CTLR.FIQEn determines whether it is signaled as a virtual IRQ or as a virtual FIQ, and GICV_CTLR.EnableGrp0 enables signaling of this interrupt to the virtual machine. |
| 0b1 | Group 1 virtual interrupt, signaled as a virtual IRQ. GICV_CTLR.EnableGrp1 enables signaling of this interrupt to the virtual machine. |

Note

[GICV_CTLR.CBPR](#) controls whether [GICV_BPR](#) or [GICV_ABPR](#) determines if a pending Group 1 interrupt has sufficient priority to preempt current execution.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

State, bits [29:28]

The state of the interrupt. This field has one of the following values:

| State | Meaning |
|-------|--------------------|
| 0b00 | Inactive |
| 0b01 | Pending |
| 0b10 | Active |
| 0b11 | Active and pending |

The GIC updates these state bits as virtual interrupts proceed through the interrupt life cycle. Entries in the inactive state are ignored, except for the purpose of generating virtual maintenance interrupts.

Note

For hardware interrupts, the active and pending state is held in the Distributor rather than the virtual CPU interface. A hypervisor must only use the active and pending state for software originated interrupts, which are typically associated with virtual devices, or for SGIs.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Priority, bits [27:23]

The priority of this interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [22:20]

Reserved, RES0.

pINTID, bits [19:10]

The function of this field depends on the value of GICH_LR<n>.HW.

When GICH_LR<n>.HW == 0:

- Bit [19] indicates whether the interrupt triggers an EOI maintenance interrupt. If this bit is 1, then when the interrupt identified by vINTID is deactivated, an EOI maintenance interrupt is asserted.
- Bits [18:13] are reserved, SBZ.

- If the vINTID field value corresponds to an SGI (that is, 0-15), bits [12:10] contain the number of the requesting PE. This appears in the corresponding field of [GICV_IAR](#) or [GICV_AIAR](#). If the vINTID field value is not 0-15, this field must be cleared to 0.

When GICH_LR<n>.HW == 1:

- This field indicates the pINTID that the hypervisor forwards to the Distributor. This field is only required to implement enough bits to hold a valid value for the ID configuration. Any unused higher order bits are RAZ/WI.
- If the value of pINTID is 0-15 or 1020-1023, behavior is UNPREDICTABLE. If the value of pINTID is 16-31, this field applies to the PPI associated with this same PE as the virtual CPU interface requesting the deactivation.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

vINTID, bits [9:0]

This INTID is returned to the VM when the interrupt is acknowledged through [GICV_IAR](#). Each valid interrupt stored in the List registers must have a unique vINTID for that virtual CPU interface. If the value of vINTID is 1020-1023, behavior is UNPREDICTABLE.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GICH_LR<n>

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICH_LR<n>](#) provides equivalent functionality.
- For AArch64 implementations, [ICH_LR<n>_EL2](#) provides equivalent functionality.

GICH_LR<n> can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------------------|---------------------------|------------|
| GIC Virtual interface control | $0 \times 0100 + (4 * n)$ | GICH_LR<n> |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

GICH_MISR, Maintenance Interrupt Status Register

The GICH_MISR characteristics are:

Purpose

Indicates which maintenance interrupts are asserted.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_MISR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICH_MISR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|--------|----|--------|----|--------|----|--------|----|-----|----|-----|----|----|----|-----|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | VGrp1D | | VGrp1E | | VGrp0D | | VGrp0E | | NPL | | REN | | U | | EOI | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

VGrp1D, bit [7]

vPE Group 1 Disabled.

| VGrp1D | Meaning |
|--------|--|
| 0b0 | vPE Group 1 Disabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 1 Disabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when [GICH_HCR.VGrp1DIE](#) == 1 and [GICH_VMCR.VENG1](#) == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp1E, bit [6]

vPE Group 1 Enabled.

| VGrp1E | Meaning |
|--------|---|
| 0b0 | vPE Group 1 Enabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 1 Enabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when [GICH_HCR.VGrp1EIE](#) == 1 and [GICH_VMCR.VENG1](#) == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp0D, bit [5]

vPE Group 0 Disabled.

| VGrp0D | Meaning |
|--------|--|
| 0b0 | vPE Group 0 Disabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 0 Disabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when [GICH_HCR.VGrp0DIE](#) == 1 and [GICH_VMCR.VENG0](#) == 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

VGrp0E, bit [4]

vPE Group 0 Enabled.

| VGrp0E | Meaning |
|--------|---|
| 0b0 | vPE Group 0 Enabled maintenance interrupt not asserted. |
| 0b1 | vPE Group 0 Enabled maintenance interrupt asserted. |

This maintenance interrupt is asserted when [GICH_HCR.VGrp0EIE](#) == 1 and [GICH_VMCR.VENG0](#) == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

NP, bit [3]

No Pending.

| NP | Meaning |
|-----|--|
| 0b0 | No Pending maintenance interrupt not asserted. |
| 0b1 | No Pending maintenance interrupt asserted. |

This maintenance interrupt is asserted when [GICH_HCR.NPIE](#) == 1 and no List register is in the pending state.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

LREN, bit [2]

List Register Entry Not Present.

| LREN | Meaning |
|------|---|
| 0b0 | List Register Entry Not Present maintenance interrupt not asserted. |
| 0b1 | List Register Entry Not Present maintenance interrupt asserted. |

This maintenance interrupt is asserted when [GICH_HCR.LRENPIE](#) == 1 and [GICH_HCR.EOICount](#) is nonzero.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

U, bit [1]

Underflow.

| U | Meaning |
|-----|---|
| 0b0 | Underflow maintenance interrupt not asserted. |
| 0b1 | Underflow maintenance interrupt asserted. |

This maintenance interrupt is asserted when [GICH_HCR](#).UIE == 1 and zero or one of the List register entries are marked as a valid interrupt.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

EOI, bit [0]

End Of Interrupt.

| EOI | Meaning |
|-----|--|
| 0b0 | End Of Interrupt maintenance interrupt not asserted. |
| 0b1 | End Of Interrupt maintenance interrupt asserted. |

This maintenance interrupt is asserted when at least one bit in [GICH_EISR](#) == 1.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Note

A List register is in the pending state only if the corresponding [GICH_LR<n>](#) value is 0b01, that is, pending. The active and pending state is not included.

Accessing GICH_MISR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICH_MISR](#) provides equivalent functionality.
- For AArch64 implementations, [ICH_MISR_EL2](#) provides equivalent functionality.

A maintenance interrupt is asserted only if at least one bit is set to 1 in this register and if [GICH_HCR](#).En == 1.

GICH_MISR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------------------|--------|-----------|
| GIC Virtual interface control | 0x0010 | GICH_MISR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

GICH_VMCR, Virtual Machine Control Register

The GICH_VMCR characteristics are:

Purpose

Enables the hypervisor to save and restore the virtual machine view of the GIC state. This register is updated when a virtual machine updates the virtual CPU interface registers.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_VMCR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICH_VMCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|----|-------|----|------|----|----|----|-------|----|------|----|-------|----|--------|----|--------|----|-------|---|-------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| VPMR | | | | VBPR0 | | VBPR1 | | RES0 | | | | VEOIM | | RES0 | | VCBPR | | VFIQEn | | VAcKct | | VENG1 | | VENG0 | | | | | | | |

VPMR, bits [31:24]

Virtual priority mask. The priority mask level for the CPU interface. If the priority of an interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

This alias field is updated when a VM updates [GICV_PMR](#).Priority.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VBPR0, bits [23:21]

Virtual Binary Point Register, Group 0. Defines the point at which the priority value fields split into two parts, the Group priority field and the subpriority field. The Group priority field determines Group 0 interrupt preemption, and also determines Group 1 interrupt preemption if GICH_VMCR.VCBPR == 1.

This alias field is updated when a VM updates [GICV_BPR](#).Binary_Point.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VBPR1, bits [20:18]

Virtual Binary Point Register, Group 1. Defines the point at which the priority value fields split into two parts, the Group priority field and the subpriority field. The Group priority field determines Group 1 interrupt preemption if GICH_VMCR.VCBPR == 0.

This alias field is updated when a VM updates [GICV_ABPR](#).Binary_Point.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [17:10]

Reserved, RES0.

VEOIM, bit [9]

Virtual EOImode. Possible values of this bit are:

| VEOIM | Meaning |
|-------|---|
| 0b0 | A write of an INTID to GICV_EOIR or GICV_AEOIR drops the priority of the interrupt with that INTID, and also deactivates that interrupt. |
| 0b1 | A write of an INTID to GICV_EOIR or GICV_AEOIR only drops the priority of the interrupt with that INTID. Software must write to GICV_DIR to deactivate the interrupt. |

This alias field is updated when a VM updates [GICV_CTLR](#).EOImode.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:5]

Reserved, RES0.

VCBPR, bit [4]

Virtual Common Binary Point Register. Possible values of this bit are:

| VCBPR | Meaning |
|-------|---|
| 0b0 | GICV_ABPR determines the preemption group for Group 1 interrupts. |
| 0b1 | GICV_BPR determines the preemption group for Group 1 interrupts. |

This alias field is updated when a VM updates [GICV_CTLR](#).CBPR.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VFIQEn, bit [3]

Virtual FIQ enable. Possible values of this bit are:

| VFIQEn | Meaning |
|--------|---|
| 0b0 | Group 0 virtual interrupts are presented as virtual IRQs. |
| 0b1 | Group 0 virtual interrupts are presented as virtual FIQs. |

This alias field is updated when a VM updates [GICV_CTLR](#).FIQEn.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VAckCtl, bit [2]

Virtual AckCtl. Possible values of this bit are:

| VAckCtl | Meaning |
|----------------|--|
| 0b0 | If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns an INTID of 1022. |
| 0b1 | If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns the INTID of the corresponding interrupt. |

This alias field is updated when a VM updates [GICV_CTLR.AckCtl](#).

This field is supported for backwards compatibility with GICv2. Arm deprecates the use of this field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VENG1, bit [1]

Virtual interrupt enable, Group 1. Possible values of this bit are:

| VENG1 | Meaning |
|--------------|--|
| 0b0 | Group 1 virtual interrupts are disabled. |
| 0b1 | Group 1 virtual interrupts are enabled. |

This alias field is updated when a VM updates [GICV_CTLR.EnableGrp1](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

VENG0, bit [0]

Virtual interrupt enable, Group 0. Possible values of this bit are:

| VENG0 | Meaning |
|--------------|--|
| 0b0 | Group 0 virtual interrupts are disabled. |
| 0b1 | Group 0 virtual interrupts are enabled. |

This alias field is updated when a VM updates [GICV_CTLR.EnableGrp0](#).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Note

A List register is in the pending state only if the corresponding [GICH_LR<n>](#) value is 0b01, that is, pending. The active and pending state is not included.

Accessing GICH_VMCR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICH_VMCR](#) provides equivalent functionality.
- For AArch64 implementations, [ICH_VMCR_EL2](#) provides equivalent functionality.

GICH_VMCR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------------------|---------------|-----------------|
| GIC Virtual interface control | 0x0008 | GICH_VMCR |

This interface is accessible as follows:

- When [GICD_CTLR.DS](#) == 0 accesses to this register are **RW**.

- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICH_VTR, Virtual Type Register

The GICH_VTR characteristics are:

Purpose

Indicates the number of implemented virtual priority bits and List registers.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICH_VTR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICH_VTR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|---------|----|----|--------|----|----|------|----|-----|------|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|----------|---|---|---|--|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
| PRIbits | | | PREbits | | | IDbits | | | SEIS | | A3V | RES0 | | | | | | | | | | | | | | | | ListRegs | | | | | |

PRIbits, bits [31:29]

The number of virtual priority bits implemented, minus one.
An implementation must implement at least 32 levels of virtual priority (5 priority bits).

PREbits, bits [28:26]

The number of virtual preemption bits implemented, minus one.
An implementation must implement at least 32 levels of virtual preemption priority (5 preemption bits).
The value of this field must be less than or equal to the value of GICH_VTR.PRIbits.

IDbits, bits [25:23]

The number of virtual interrupt identifier bits supported:

| IDbits | Meaning |
|--------|----------|
| 0b000 | 16 bits. |
| 0b001 | 24 bits. |

All other values are reserved.

SEIS, bit [22]

SEI support. Indicates whether the virtual CPU interface supports generation of SEIs:

| SEIS | Meaning |
|------|--|
| 0b0 | The virtual CPU interface logic does not support generation of SEIs. |
| 0b1 | The virtual CPU interface logic supports generation of SEIs. |

A3V, bit [21]

Affinity 3 valid. Possible values are:

| A3V | Meaning |
|------------|---|
| 0b0 | The virtual CPU interface logic only supports zero values of the Aff3 field in ICC_SGI0R_EL1 , ICC_SGI1R_EL1 , and ICC_ASGI1R_EL1 . |
| 0b1 | The virtual CPU interface logic supports nonzero values of the Aff3 field in ICC_SGI0R_EL1 , ICC_SGI1R_EL1 , and ICC_ASGI1R_EL1 . |

Bits [20:5]

Reserved, RES0.

ListRegs, bits [4:0]

The number of implemented List registers, minus one.

Accessing GICH_VTR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICH_VTR](#) provides equivalent functionality.
- For AArch64 implementations, [ICH_VTR_EL2](#) provides equivalent functionality.

GICH_VTR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-------------------------------|---------------|-----------------|
| GIC Virtual interface control | 0x0004 | GICH_VTR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICM_CLRSPI_NSR, Clear Non-secure SPI Pending Register

The GICM_CLRSPI_NSR characteristics are:

Purpose

Removes the pending state from a valid SPI if permitted by the Security state of the access and the [GICD_NSACR<n>](#) value for that SPI.

A write to this register changes the state of a pending SPI to inactive, and the state of an active and pending SPI to active.

Configuration

This register is present only when GICM_TYPER.CLR == 1. Otherwise, direct accesses to GICM_CLRSPI_NSR are RES0.

When [GICD_CTLR.DS](#) == 1, this register provides functionality for all SPIs.

Attributes

GICM_CLRSPI_NSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |

Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

This field is an alias of [GICD_CLRSPI_NSR](#).

Accessing GICM_CLRSPI_NSR

Writes to this register have no effect if:

- The value written specifies a Secure SPI, the value is written by a Non-secure access, and the value of the corresponding [GICD_NSACR<n>](#) register is less than 0b10.
- The value written specifies an invalid SPI.
- The SPI is not pending.

16-bit accesses to bits [15:0] of this register must be supported.

Note

A Secure access to this register can clear the pending state of any valid SPI.

GICM_CLRSPI_NSR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|----------|--------|-----------------|
| GIC Distributor | MSI_base | 0x0048 | GICM_CLRSPI_NSR |

Accesses on this interface are **WO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICM_CLRSPI_SR, Clear Secure SPI Pending Register

The GICM_CLRSPI_SR characteristics are:

Purpose

Removes the pending state from a valid SPI.

A write to this register changes the state of a pending SPI to inactive, and the state of an active and pending SPI to active.

Configuration

This register is present only when GICM_TYPER.SR == 1 and GICM_TYPER.CLR == 1. Otherwise, direct accesses to GICM_CLRSPI_SR are RES0.

When [GICD_CTLR.DS](#) == 1, this register is **WI**.

Attributes

GICM_CLRSPI_SR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |

Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

This field is an alias of [GICD_CLRSPI_SR](#).

Accessing GICM_CLRSPI_SR

Writes to this register have no effect if:

- The value is written by a Non-secure access.
- The value written specifies an invalid SPI.
- The SPI is not pending.

16-bit accesses to bits [15:0] of this register must be supported.

GICM_CLRSPI_SR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|----------|--------|----------------|
| GIC Distributor | MSI_base | 0x0058 | GICD_CLRSPI_SR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 1 accesses to this register are **WI**.
- When GICD_CTLR.DS == 0 and an access is Secure accesses to this register are **WO**.
- When GICD_CTLR.DS == 0 and an access is Non-secure accesses to this register are **WI**.

- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRoot() accesses to this register are **WO**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRealm() accesses to this register are **WI**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICM_IIDR, Distributor Implementer Identification Register

The GICM_IIDR characteristics are:

Purpose

Provides information about the implementer and revision of the Distributor.

Configuration

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states, this register is Common.

Attributes

GICM_IIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|------|----|----|----|---------|----|----|----|----------|----|----|----|-------------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ProductID | | | | | | | | RES0 | | | | Variant | | | | Revision | | | | Implementer | | | | | | | | | | | |

ProductID, bits [31:24]

Product Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [23:20]

Reserved, RES0.

Variant, bits [19:16]

Variant number. Typically, this field is used to distinguish product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

Revision number. Typically, this field is used to distinguish minor revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 code of the company that implemented the Distributor:

- Bits [11:8] are the JEP106 continuation code of the implementer. For an Arm implementation, this field is 0x4.
- Bit [7] is always 0.
- Bits [6:0] are the JEP106 identity code of the implementer. For an Arm implementation, bits [7:0] are therefore 0x3B.

Accessing GICM_IIDR

GICM_IIDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|--------------------|----------|--------|-----------|
| GIC Distributor | MSI_base | 0x0FCC | GICM_IIDR |

Accesses on this interface are **RO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICM_SETSPI_NSR, Set Non-secure SPI Pending Register

The GICM_SETSPI_NSR characteristics are:

Purpose

Adds the pending state to a valid SPI if permitted by the Security state of the access and the [GICD_NSACR<n>](#) value for that SPI.

A write to this register changes the state of an inactive SPI to pending, and the state of an active SPI to active and pending.

Configuration

When [GICD_CTLR.DS](#)==1, this register provides functionality for all SPIs.

Attributes

GICM_SETSPI_NSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |

Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

This field is an alias of [GICD_SETSPI_NSR](#).

Accessing GICM_SETSPI_NSR

Writes to this register have no effect if:

- The value written specifies a Secure SPI, the value is written by a Non-secure access, and the value of the corresponding [GICD_NSACR<n>](#) register is 0.
- The value written specifies an invalid SPI.
- The SPI is already pending.

16-bit accesses to bits [15:0] of this register must be supported.

Note

A Secure access to this register can set the pending state of any valid SPI.

GICM_SETSPI_NSR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|----------|--------|-----------------|
| GIC Distributor | MSI_base | 0x0040 | GICM_SETSPI_NSR |

Accesses on this interface are **WO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICM_SETSPI_SR, Set Secure SPI Pending Register

The GICM_SETSPI_SR characteristics are:

Purpose

Adds the pending state to a valid SPI.

A write to this register changes the state of an inactive SPI to pending, and the state of an active SPI to active and pending.

Configuration

This register is present only when GICM_TYPER.SR == 1. Otherwise, direct accesses to GICM_SETSPI_SR are RES0.

When [GICD_CTLR.DS](#)==1, this register is **WI**.

Attributes

GICM_SETSPI_SR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | INTID | | | | | | | | | | | | | | | |

Bits [31:13]

Reserved, RES0.

INTID, bits [12:0]

This field is an alias of [GICD_SETSPI_SR](#).

Accessing GICM_SETSPI_SR

Writes to this register have no effect if:

- The value is written by a Non-secure access.
- The value written specifies an invalid SPI.
- The SPI is already pending.

16-bit accesses to bits [15:0] of this register must be supported.

GICM_SETSPI_SR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------------|----------|--------|----------------|
| GIC Distributor | MSI_base | 0x0050 | GICM_SETSPI_SR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 1 accesses to this register are **WI**.
- When GICD_CTLR.DS == 0 and an access is Secure accesses to this register are **WO**.
- When GICD_CTLR.DS == 0 and an access is Non-secure accesses to this register are **WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRoot() accesses to this register are **WO**.

- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRealm() accesses to this register are **WI**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICM_TYPER, Distributor MSI Type Register

The GICM_TYPER characteristics are:

Purpose

Provides information about what features the GIC implementation supports.

Configuration

This register is available in all configurations of the GIC. When [GICD_CTLR.DS](#)=0, this register is Common.

Attributes

GICM_TYPER is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|-----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|---------|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Valid | CLR | SR | INTID | | | | | | | | | | | | | RES0 | | | | | NumSPis | | | | | | | | | | |

Valid, bit [31]

Reports whether GICM_TYPER content is valid.

| Valid | Meaning |
|-------|---|
| 0b0 | GICM_TYPER reports no information on the capabilities of the GICM frame, all other fields are RES0. |
| 0b1 | GICM_TYPER reports information on capabilities of GICM frame. |

CLR, bit [30]

Reports whether MSI clear registers are supported.

| CLR | Meaning |
|-----|--------------------------------------|
| 0b0 | MSI clear registers not implemented. |
| 0b1 | MSI clear registers implemented. |

SR, bit [29]

Reports whether Secure aliases of MSI registers are supported.

| SR | Meaning |
|-----|--|
| 0b0 | Secure aliases of MSI registers not implemented. |
| 0b1 | Secure aliases of MSI registers implemented. |

INTID, bits [28:16]

INTID of the first SPI assigned to this GICM frame.

Bits [15:11]

Reserved, RES0.

NumSPIs, bits [10:0]

Number of SPIs assigned to this GICM frame.

Accessing GICM_TYPER

GICM_TYPER can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|--------------------|----------|--------|------------|
| GIC Distributor | MSI_base | 0x0004 | GICM_TYPER |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_CLRLPIR, Clear LPI Pending Register

The GICR_CLRLPIR characteristics are:

Purpose

Clears the pending state of the specified LPI.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_CLRLPIR is a 64-bit register.

Field descriptions

When GICR_TYPER.DirectLPI == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | pINTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

pINTID, bits [31:0]

The INTID of the physical LPI.

Note

The size of this field is IMPLEMENTATION DEFINED, and is specified by the [GICD_TYPER.IDbits](#) field. Unimplemented bits are RES0.

When GICR_TYPER.DirectLPI == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing GICR_CLRLPIR

When written with a 32-bit write the data is zero-extended to 64 bits.

This register is mandatory in an implementation that supports LPIs and does not include an ITS. The functionality of this register is IMPLEMENTATION DEFINED in an implementation that does include an ITS.

Writes to this register have no effect if any of the following apply:

- [GICR_CTLR](#).EnableLPIS == 0.
- The pINTID value specifies an unimplemented LPI.
- The pINTID value specifies an LPI that is not pending.

GICR_CLRLPIR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|----------------------|---------|--------|--------------|
| GIC Redistributor | RD_base | 0x0048 | GICR_CLRLPIR |

Accesses on this interface are **WO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_CTLR, Redistributor Control Register

The GICR_CTLR characteristics are:

Purpose

Controls the operation of a Redistributor, and enables the signaling of LPIs by the Redistributor to the connected PE.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_CTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|------|-------|--------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| UWP | RES0 | DPG1S | DPG1NS | DPG0 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

UWP, bit [31]

Upstream Write Pending. Read-only. Indicates whether all upstream writes have been communicated to the Distributor.

| UWP | Meaning |
|-----|---|
| 0b0 | The effects of all upstream writes have been communicated to the Distributor, including any Generate SGI packets. For more information, see 'Generate SGI (ICC)' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069). |
| 0b1 | Not all the effects of upstream writes, including any Generate SGI packets, have been communicated to the Distributor. For more information, see 'Generate SGI (ICC)' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069). |

Bits [30:27]

Reserved, RES0.

DPG1S, bit [26]

Disable Processor selection for Group 1 Secure interrupts. When [GICR_TYPER](#).DPGS == 1:

| DPG1S | Meaning |
|-------|--|
| 0b0 | A Group 1 Secure SPI configured to use the 1 of N distribution model can select this PE, if the PE is not asleep and if Secure Group 1 interrupts are enabled. |
| 0b1 | A Group 1 Secure SPI configured to use the 1 of N distribution model cannot select this PE. |

When [GICR_TYPER](#).DPGS == 0 this bit is RAZ/WI.

When [GICD_CTLR](#).DS==1, this field is RAZ/WI. In GIC implementations that support two Security states, this field is only accessible by Secure accesses, and is RAZ/WI to Non-secure accesses.

It is IMPLEMENTATION DEFINED whether these bits affect the selection of PEs for interrupts using the 1 of N distribution model when [GICD_CTLR.ARE_S](#)==0.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

DPG1NS, bit [25]

Disable Processor selection for Group 1 Non-secure interrupts. When [GICR_TYPER.DPGS](#) == 1:

| DPG1NS | Meaning |
|--------|--|
| 0b0 | A Group 1 Non-secure SPI configured to use the 1 of N distribution model can select this PE, if the PE is not asleep and if Non-secure Group 1 interrupts are enabled. |
| 0b1 | A Group 1 Non-secure SPI configured to use the 1 of N distribution model cannot select this PE. |

When [GICR_TYPER.DPGS](#) == 0 this bit is RAZ/WI.

It is IMPLEMENTATION DEFINED whether these bits affect the selection of PEs for interrupts using the 1 of N distribution model when [GICD_CTLR.ARE_NS](#)==0.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

DPG0, bit [24]

Disable Processor selection for Group 0 interrupts. When [GICR_TYPER.DPGS](#) == 1:

| DPG0 | Meaning |
|------|--|
| 0b0 | A Group 0 SPI configured to use the 1 of N distribution model can select this PE, if the PE is not asleep and if Group 0 interrupts are enabled. |
| 0b1 | A Group 0 SPI configured to use the 1 of N distribution model cannot select this PE. |

When [GICR_TYPER.DPGS](#) == 0 this bit is RAZ/WI.

When [GICD_CTLR.DS](#) == 1, this field is always accessible. In GIC implementations that support two Security states, this field is RAZ/WI to Non-secure accesses.

It is IMPLEMENTATION DEFINED whether these bits affect the selection of PEs for interrupts using the 1 of N distribution model when [GICD_CTLR.ARE_S](#) == 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Bits [23:4]

Reserved, RES0.

RWP, bit [3]

Register Write Pending. This bit indicates whether a register write for the current Security state is in progress or not.

| RWP | Meaning |
|-----|--|
| 0b0 | The effect of all previous writes to the following registers are visible to all agents in the system: <ul style="list-style-type: none"> GICR_ICENABLER0 GICR_CTLR.DPG1S GICR_CTLR.DPG1NS GICR_CTLR.DPG0 GICR_CTLR, which clears EnableLPis from 1 to 0. In FEAT_GICv4p1, GICR_VPROPBASER, which clears Valid from 1 to 0. |
| 0b1 | The effect of all previous writes to the following registers are not guaranteed by the architecture to be visible to all agents in the system while the changes are still being propagated: <ul style="list-style-type: none"> GICR_ICENABLER0 GICR_CTLR.DPG1S GICR_CTLR.DPG1NS GICR_CTLR.DPG0 GICR_CTLR, which clears EnableLPis from 1 to 0. In FEAT_GICv4p1, GICR_VPROPBASER, which clears Valid from 1 to 0. |

IR, bit [2]

LPI invalidate registers supported.

This bit is read-only.

| IR | Meaning |
|-----|---|
| 0b0 | This bit does not indicate whether the GICR_INVLPIR, GICR_INVALLR and GICR_SYNCRR are implemented or not. |
| 0b1 | GICR_INVLPIR, GICR_INVALLR and GICR_SYNCRR are implemented. |

If [GICR_TYPER.DirectLPI](#) is 1 or [GICR_TYPER.RVPEI](#) is 1, [GICR_INVLPIR](#), [GICR_INVALLR](#), and [GICR_SYNCRR](#) are always implemented.

Arm recommends that implementations report GICR_CTLR.IR as 1 in these cases.

CES, bit [1]

Clear Enable Supported.

This bit is read-only.

| CES | Meaning |
|-----|--|
| 0b0 | The IRI does not indicate whether GICR_CTLR.EnableLPis is RES1 once set. |
| 0b1 | GICR_CTLR.EnableLPis is not RES1 once set. |

Implementing GICR_CTLR.EnableLPis as programmable and not reporting GICR_CTLR.CES == 1 is deprecated.

Implementing GICR_CTLR.EnableLPis as RES1 once set is deprecated.

When GICR_CTLR.CES == 0, software cannot assume that GICR_CTLR.EnableLPis is programmable without observing the bit being cleared.

EnableLPis, bit [0]

In implementations where affinity routing is enabled for the Security state:

| EnableLPis | Meaning |
|------------|--|
| 0b0 | LPI support is disabled. Any doorbell interrupt generated as a result of a write to a virtual LPI register must be discarded, and any ITS translation requests or commands involving LPis in this Redistributor are ignored. |
| 0b1 | LPI support is enabled. |

Note

If [GICR_TYPER](#).PLPIS == 0, this field is RES0. If [GICD_CTLR](#).ARE_NS is written from 1 to 0 when this bit is 1, behavior is an IMPLEMENTATION DEFINED choice between clearing GICR_CTLR.EnableLPis to 0 or maintaining its current value.

When affinity routing is not enabled for the Non-secure state, this bit is RES0.

When written from 0 to 1, the Redistributor loads the LPI Pending table from memory to check for any pending interrupts.

After it has been written to 1, it is IMPLEMENTATION DEFINED whether the bit becomes RES1 or can be cleared by to 0.

Where the bit remains programmable:

- Software must observe GICR_CTLR.RWP==0 after clearing GICR_CTLR.EnableLPis from 1 to 0 before writing [GICR_PENDBASER](#) or [GICR_PROPBASER](#), otherwise behavior is UNPREDICTABLE.
- Software must observe GICR_CTLR.RWP==0 after clearing GICR_CTLR.EnableLPis from 1 to 0 before setting GICR_CTLR.EnableLPis to 1, otherwise behavior is UNPREDICTABLE.

Note

If one or more ITS is implemented, Arm strongly recommends that all LPis are mapped to another Redistributor before GICR_CTLR.EnableLPis is cleared to 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

The participation of a PE in the 1 of N distribution model for a given interrupt group is governed by the concatenation of [GICR_WAKER](#).ProcessorSleep, the appropriate [GICR_CTLR](#).DPG{1, 0} bit, and the PE interrupt group enable. The behavior options are:

| PS | DPG{1S, 1NS, 0} | Enable | PE Behavior |
|-----|-----------------|--------|--|
| 0b0 | 0b0 | 0b0 | The PE cannot be selected. |
| 0b0 | 0b0 | 0b1 | The PE can be selected. |
| 0b0 | 0b1 | * | The PE cannot be selected. |
| 0b1 | * | * | The PE cannot be selected when GICD_CTLR .E1NWF == 0. When GICD_CTLR .E1NWF == 1, the mechanism by which PEs are selected is IMPLEMENTATION DEFINED. |

If an SPI using the 1 of N distribution model has been forwarded to the PE, and a write to GICR_CTLR occurs that changes the DPG bit for the interrupt group of the SPI, the IRI must attempt to select a different target PE for the SPI. This might have no effect on the forwarded SPI if it has already been activated.

Accessing GICR_CTLR

GICR_CTLR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|---------|--------|-----------|
| GIC Redistributor | RD_base | 0x0000 | GICR_CTLR |

Accesses on this interface are **RW**.

GICR_ICACTIVER0, Interrupt Clear-Active Register 0

The GICR_ICACTIVER0 characteristics are:

Purpose

Deactivates the corresponding SGI or PPI. These registers are used when saving and restoring GIC state.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICACTIVER0 is a 32-bit register.

Field descriptions

| | | | | | | |
|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | Cle |
| Clear_active_bit31 | Clear_active_bit30 | Clear_active_bit29 | Clear_active_bit28 | Clear_active_bit27 | Clear_active_bit26 | Clear_active_bit25 |

Clear_active_bit<x>, bit [x], for x = 31 to 0

Removes the active state from interrupt number x. Reads and writes have the following behavior:

| Clear_active_bit<x> | Meaning |
|---------------------|--|
| 0b0 | If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is active, or is active and pending. If written, deactivates the corresponding interrupt, if the interrupt is active. If the interrupt is already deactivated, the write has no effect. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_ICACTIVER0

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICACTIVER0, the corresponding bit is RAZ/WI and equivalent functionality is provided by [GICD_ICACTIVER<n>](#) with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by [GICD_ICACTIVER<n>](#).

When [GICD_CTLR](#).DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

GICR_ICACTIVER0 can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|--------|-----------------|
| GIC Redistributor | SGI_base | 0x0380 | GICR_ICACTIVER0 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICACTIVER<n>E, Interrupt Clear-Active Registers, n = 1 - 2

The GICR_ICACTIVER<n>E characteristics are:

Purpose

Removes the active state from the corresponding PPI.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICR_ICACTIVER<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICACTIVER<n>E is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Clear_active_bit31 | Clear_active_bit30 | Clear_active_bit29 | Clear_active_bit28 | Clear_active_bit27 | Clear_active_bit26 | Clear_active_bit25 | Clear_active_bit24 | Clear_active_bit23 | Clear_active_bit22 | Clear_active_bit21 | Clear_active_bit20 | Clear_active_bit19 | Clear_active_bit18 | Clear_active_bit17 | Clear_active_bit16 | Clear_active_bit15 | Clear_active_bit14 | Clear_active_bit13 | Clear_active_bit12 | Clear_active_bit11 | Clear_active_bit10 | Clear_active_bit9 | Clear_active_bit8 | Clear_active_bit7 | Clear_active_bit6 | Clear_active_bit5 | Clear_active_bit4 | Clear_active_bit3 | Clear_active_bit2 | Clear_active_bit1 | Clear_active_bit0 |

Clear_active_bit<x>, bit [x], for x = 31 to 0

For the extended PPIs, removes the active state to interrupt number x. Reads and writes have the following behavior:

| Clear_active_bit<x> | Meaning |
|---------------------|--|
| 0b0 | If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is active, or is active and pending. If written, deactivates the corresponding interrupt, if the interrupt is active. If the interrupt is already deactivated, the write has no effect. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ICACTIVER<n>E number, n, is given by $n = (m-1024) \text{ DIV } 32$.
- The offset of the required GICR_ICACTIVER<n>E is $(0x200 + (4*n))$.
- The bit number of the required group modifier bit in this register is $(m-1024) \text{ MOD } 32$.

Accessing GICR_ICACTIVER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICACTIVER<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)=0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_ICACTIVER<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|------------------------|--------------------|
| GIC Redistributor | SGI_base | 0x0380 + (4 * n) | GICR_ICACTIVER<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICENABLER0, Interrupt Clear-Enable Register 0

The GICR_ICENABLER0 characteristics are:

Purpose

Disables forwarding of the corresponding SGI or PPI to the CPU interfaces.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICENABLER0 is a 32-bit register.

Field descriptions

| | | | | | |
|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 |
| Clear_enable_bit31 | Clear_enable_bit30 | Clear_enable_bit29 | Clear_enable_bit28 | Clear_enable_bit27 | Clear_enable_bit26 |

Clear_enable_bit<x>, bit [x], for x = 31 to 0

For PPIs and SGIs, controls the forwarding of interrupt number x to the CPU interfaces. Reads and writes have the following behavior:

| Clear_enable_bit<x> | Meaning |
|---------------------|--|
| 0b0 | If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect. |
| 0b1 | If read, indicates that forwarding of the corresponding interrupt is enabled. If written, disables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 0. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_ICENABLER0

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICENABLER0, the corresponding bit is RAZ/WI and equivalent functionality is provided by [GICD_ICENABLER<n>](#) with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by [GICD_ICENABLER<n>](#).

When [GICD_CTLR](#).DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

GICR_ICENABLER0 can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|--------|-----------------|
| GIC Redistributor | SGI_base | 0x0180 | GICR_ICENABLER0 |

Accesses on this interface are **RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICENABLER<n>E, Interrupt Clear-Enable Registers, n = 1 - 2

The GICR_ICENABLER<n>E characteristics are:

Purpose

Disables forwarding of the corresponding PPI to the CPU interfaces.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICR_ICENABLER<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICENABLER<n>E is a 32-bit register.

Field descriptions

| | | | | | |
|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 |
| Clear_enable_bit31 | Clear_enable_bit30 | Clear_enable_bit29 | Clear_enable_bit28 | Clear_enable_bit27 | Clear_enable_bit26 |

Clear_enable_bit<x>, bit [x], for x = 31 to 0

For the extended PPI range, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

| Clear_enable_bit<x> | Meaning |
|---------------------|--|
| 0b0 | If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect. |
| 0b1 | If read, indicates that forwarding of the corresponding interrupt is enabled. If written, disables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 0. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ICENABLER<n>E number, n, is given by $n = (m-1024) \text{ DIV } 32$.
- The offset of the required GICR_ICENABLER<n>E is $(0x180 + (4*n))$.
- The bit number of the required group modifier bit in this register is $(m-1024) \text{ MOD } 32$.

Accessing GICR_ICENABLER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICENABLER<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)=0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_ICENABLER<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|------------------------|--------------------|
| GIC Redistributor | SGI_base | 0x0180 + (4 * n) | GICR_ICENABLER<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICFGR0, Interrupt Configuration Register 0

The GICR_ICFGR0 characteristics are:

Purpose

Determines whether the corresponding SGI is edge-triggered or level-sensitive.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICFGR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | |
|------------------------------|------------------------------|------------------------------|------------------------------|------------------------------|------------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 |
| Int_config15 | Int_config14 | Int_config13 | Int_config12 | Int_config11 | Int_config10 | Int_config9 | Int_config8 | Int_config7 | Int_config6 | Int_config5 | Int_config4 | Int_config3 | Int_config2 | Int_config1 | Int_config0 | Int_config0 | Int_config0 | Int_config0 |

Int_config<x>, bits [2x+1:2x], for x = 15 to 0

Indicates whether the is level-sensitive or edge-triggered.

| Int_config<x> | Meaning |
|---------------|---|
| 0b00 | Corresponding interrupt is level-sensitive. |
| 0b10 | Corresponding interrupt is edge-triggered. |

SGIs are always edge-triggered.

When the interrupt is visible to the current Security state, a read of this bit always returns the correct value to indicate the interrupt triggering method.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_ICFGR0

This register is used when affinity routing is enabled.

When affinity routing is disabled for the Security state of an interrupt, the field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case. Equivalent functionality is provided by GICD_ICFGR<n> with n=0.

When [GICD_CTLR.DS](#)=0, a register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

GICR_ICFGR0 can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|--------|-------------|
| GIC Redistributor | SGI_base | 0x0C00 | GICR_ICFGR0 |

Accesses on this interface are **RW**.

GICR_ICFGR1, Interrupt Configuration Register 1

The GICR_ICFGR1 characteristics are:

Purpose

Determines whether the corresponding PPI is edge-triggered or level-sensitive.

Configuration

A copy of this register is provided for each Redistributor.

For each supported PPI, it is IMPLEMENTATION DEFINED whether software can program the corresponding Int_config field.

Changing Int_config when the interrupt is individually enabled is UNPREDICTABLE.

Changing the interrupt configuration between level-sensitive and edge-triggered (in either direction) at a time when there is a pending interrupt will leave the interrupt in an UNKNOWN pending state.

Attributes

GICR_ICFGR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | |
|------------------------------|------------------------------|------------------------------|------------------------------|------------------------------|------------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 |
| Int_config15 | Int_config14 | Int_config13 | Int_config12 | Int_config11 | Int_config10 | Int_config9 | Int_config8 | Int_config7 | Int_config6 | Int_config5 | Int_config4 | Int_config3 | Int_config2 | Int_config1 | Int_config0 | Int_config0 | Int_config0 | Int_config0 |

Int_config<x>, bits [2x+1:2x], for x = 15 to 0

Indicates whether the interrupt is level-sensitive or edge-triggered.

| Int_config<x> | Meaning |
|---------------|---|
| 0b00 | Corresponding interrupt is level-sensitive. |
| 0b10 | Corresponding interrupt is edge-triggered. |

Int_config[0] (bit [2x]) is RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_ICFGR1

This register is used when affinity routing is enabled.

When affinity routing is disabled for the Security state of an interrupt, the field for that interrupt is RES0 and an implementation is permitted to make the field RAZ/WI in this case. Equivalent functionality is provided by GICD_ICFGR<n> with n=1 .

When [GICD_CTLR.DS](#)=0, a register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

GICR_ICFGR1 can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|-------|--------|----------|
|-----------|-------|--------|----------|

GICR_ICFGR1, Interrupt Configuration Register 1

| | | | |
|----------------------|----------|--------|-------------|
| GIC Redistributor | SGI_base | 0x0C04 | GICR_ICFGR1 |
|----------------------|----------|--------|-------------|

Accesses on this interface are **RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICFGR<n>E, Interrupt configuration registers, n = 2 - 5

The GICR_ICFGR<n>E characteristics are:

Purpose

Determines whether the corresponding PPI in the extended PPI range is edge-triggered or level-sensitive.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICR_ICFGR<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICFGR<n>E is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | |
|--------------|--------------|--------------|--------------|--------------|--------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|-------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 |
| Int_config15 | Int_config14 | Int_config13 | Int_config12 | Int_config11 | Int_config10 | Int_config9 | Int_config8 | Int_config7 | Int_config6 | Int_config5 | Int_config4 | Int_config3 | Int_config2 | Int_config1 | Int_config0 | Int_config0 | Int_config0 | Int_config0 |

Int_config<x>, bits [2x+1:2x], for x = 15 to 0

Indicates whether the interrupt is level-sensitive or edge-triggered.

Int_config[0] (bit [2x]) is RES0.

| Int_config<x> | Meaning |
|---------------|---|
| 0b00 | The corresponding interrupt is level-sensitive. |
| 0b10 | The corresponding interrupt is edge-triggered. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

For each supported extended PPI, it is IMPLEMENTATION DEFINED whether software can program the corresponding Int_config field.

Accessing GICR_ICFGR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICFGR<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)=0, a register bit that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_ICFGR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|-------|--------|----------|
|-----------|-------|--------|----------|

| | | | |
|----------------------|----------|---------------------|----------------|
| GIC Redistributor | SGI_base | 0x0C00 + (4 * n) | GICR_ICFGR<n>E |
|----------------------|----------|---------------------|----------------|

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICPENDR0, Interrupt Clear-Pending Register 0

The GICR_ICPENDR0 characteristics are:

Purpose

Removes the pending state from the corresponding SGI or PPI.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICPENDR0 is a 32-bit register.

Field descriptions

| | | | | | |
|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 |
| Clear_pending_bit31 | Clear_pending_bit30 | Clear_pending_bit29 | Clear_pending_bit28 | Clear_pending_bit27 | Clear_pending_bit26 |

Clear_pending_bit<x>, bit [x], for x = 31 to 0

Removes the pending state from interrupt number x. Reads and writes have the following behavior:

| Clear_pending_bit<x> | Meaning |
|----------------------|--|
| 0b0 | If read, indicates that the corresponding interrupt is not pending. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is pending, or active and pending. If written, changes the state of the corresponding interrupt from pending to inactive, or from active and pending to active. This has no effect in the following cases: <ul style="list-style-type: none"> If the interrupt is not pending and is not active and pending. If the interrupt is a level-sensitive interrupt that is pending or active and pending for a reason other than a write to GICD_ISPENDR<n>. In this case, if the interrupt signal continues to be asserted, the interrupt remains pending or active and pending. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_ICPENDR0

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICPENDR0, the corresponding bit is RAZ/WI and equivalent functionality is provided by [GICD_ICPENDR<n>](#) with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by [GICD_ICENABLER<n>](#).

When [GICD_CTLR.DS](#) == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

GICR_ICPENDR0 can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|----------------------|----------|--------|---------------|
| GIC Redistributor | SGI_base | 0x0280 | GICR_ICPENDR0 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ICPENDR<n>E, Interrupt Clear-Pending Registers, n = 1 - 2

The GICR_ICPENDR<n>E characteristics are:

Purpose

Removes the pending state from the corresponding PPI.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICR_ICPENDR<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ICPENDR<n>E is a 32-bit register.

Field descriptions

| | | | | | |
|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|-------------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 |
| Clear_pending_bit31 | Clear_pending_bit30 | Clear_pending_bit29 | Clear_pending_bit28 | Clear_pending_bit27 | Clear_pending_bit26 |

Clear_pending_bit<x>, bit [x], for x = 31 to 0

For the extended PPIs, removes the pending state to interrupt number x. Reads and writes have the following behavior:

| Clear_pending_bit<x> | Meaning |
|----------------------|--|
| 0b0 | If read, indicates that the corresponding interrupt is not pending on this PE. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is pending, or active and pending on this PE. If written, changes the state of the corresponding interrupt from pending to inactive, or from active and pending to active. This has no effect in the following cases: <ul style="list-style-type: none">• If the interrupt is not pending and is not active and pending.• If the interrupt is a level-sensitive interrupt that is pending or active and pending for a reason other than a write to GICR_ICPENDR<n>E. In this case, if the interrupt signal continues to be asserted, the interrupt remains pending or active and pending. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ICPENDR<n>E number, n, is given by $n = (m-1024) \text{ DIV } 32$.
- The offset of the required GICR_ICPENDR<n>E is $(0x200 + (4*n))$.
- The bit number of the required group modifier bit in this register is $(m-1024) \text{ MOD } 32$.

Accessing GICR_ICPENDR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ICPENDR<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)=0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_ICPENDR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|--------------------|------------------|
| GIC Redistributor | SGI_base | $0x0280 + (4 * n)$ | GICR_ICPENDR<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_IGROUPR0, Interrupt Group Register 0

The GICR_IGROUPR0 characteristics are:

Purpose

Controls whether the corresponding SGI or PPI is in Group 0 or Group 1.

Configuration

This register is available in all GIC configurations. If the GIC implementation supports two Security states, this register is Secure.

A copy of this register is provided for each Redistributor.

Attributes

GICR_IGROUPR0 is a 32-bit register.

Field descriptions

| | | | |
|--|--|--|--|
| 31 | 30 | 29 | |
| Redistributor_group_status_bit31 | Redistributor_group_status_bit30 | Redistributor_group_status_bit29 | Redistributor_group_status_bit28 |

Redistributor_group_status_bit<x>, bit [x], for x = 31 to 0

Group status bit. In this register:

- Bits [31:16] are group status bits for PPIs.
- Bits [15:0] are group status bits for SGIs.

| Redistributor_group_status_bit<x> | Meaning |
|-----------------------------------|---|
| 0b0 | When GICD_CTLR.DS ==1, the corresponding interrupt is Group 0. When GICD_CTLR.DS ==0, the corresponding interrupt is Secure. |
| 0b1 | When GICD_CTLR.DS ==1, the corresponding interrupt is Group 1. When GICD_CTLR.DS ==0, the corresponding interrupt is Non-secure Group 1. |

When [GICD_CTLR.DS](#) == 0, the bit that corresponds to the interrupt is concatenated with the equivalent bit in [GICR_IGRPMODR0](#) to form a 2-bit field that defines an interrupt group. The encoding of this field is at [GICR_IGRPMODR0](#).

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

The considerations for the reset value of this register are the same as those for [GICD_IGROUPR<n>](#) with n=0.

Accessing GICR_IGROUPR0

When affinity routing is not enabled for the Security state of an interrupt in GICR_IGROUPR0, the corresponding bit is RES0 and equivalent functionality is provided by [GICD_IGROUPR<n>](#) with n=0.

When [GICD_CTLR.DS](#) == 0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICR_IGROUPR0 can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|--------|---------------|
| GIC Redistributor | SGI_base | 0x0080 | GICR_IGROUPR0 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_IGROUPR<n>E, Interrupt Group Registers, n = 1 - 2

The GICR_IGROUPR<n>E characteristics are:

Purpose

Controls whether the corresponding PPI is in Group 0 or Group 1.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICR_IGROUPR<n>E are RES0.

When [GICD_CTLR.DS](#)==0, this register is Secure.

A copy of this register is provided for each Redistributor.

Attributes

GICR_IGROUPR<n>E is a 32-bit register.

Field descriptions

| | | | | | |
|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|------------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 |
| Group_status_bit31 | Group_status_bit30 | Group_status_bit29 | Group_status_bit28 | Group_status_bit27 | Group_status_bit26 |

Group_status_bit<x>, bit [x], for x = 31 to 0

Group status bit.

| Group_status_bit<x> | Meaning |
|---------------------|---|
| 0b0 | When GICD_CTLR.DS ==1, the corresponding interrupt is Group 0. When GICD_CTLR.DS ==0, the corresponding interrupt is Secure. |
| 0b1 | When GICD_CTLR.DS ==1, the corresponding interrupt is Group 1. When GICD_CTLR.DS ==0, the corresponding interrupt is Non-secure Group 1. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

If affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in GICR_IGRPMODR<n>E to form a 2-bit field that defines an interrupt group. The encoding of this field is described in GICR_IGRPMODR<n>E.

If affinity routing is disabled for the Security state of an interrupt, the bit is RES0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_IGROUPR<n>E number, n, is given by $n = (m-1024) \text{ DIV } 32$.
- The offset of the required GICR_IGROUPR<n>E is $(0x080 + (4*n))$.
- The bit number of the required group modifier bit in this register is $(m-1024) \text{ MOD } 32$.

Accessing GICR_IGROUPR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_IGROUPR<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)==0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_IGROUPR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|------------------------|------------------|
| GIC Redistributor | SGI_base | 0x0080 + (4 * n) | GICR_IGROUPR<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_IGRPMODR0, Interrupt Group Modifier Register 0

The GICR_IGRPMODR0 characteristics are:

Purpose

When [GICD_CTLR.DS](#)==0, this register together with the [GICR_IGROUPR0](#) register, controls whether the corresponding interrupt is in:

- Secure Group 0.
- Non-secure Group 1.
- When System register access is enabled, Secure Group 1.

Configuration

When [GICD_CTLR.DS](#)==0, this register is Secure.

A copy of this register is provided for each Redistributor.

Attributes

GICR_IGRPMODR0 is a 32-bit register.

Field descriptions

| | | | | | |
|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|
| 31 | 30 | 29 | 28 | 27 | |
| Group_modifier_bit31 | Group_modifier_bit30 | Group_modifier_bit29 | Group_modifier_bit28 | Group_modifier_bit27 | Group_modifier_bit26 |

Group_modifier_bit<x>, bit [x], for x = 31 to 0

Group modifier bit. In implementations where affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in [GICR_IGROUPR0](#) to form a 2-bit field that defines an interrupt group:

| Group modifier bit | Group status bit | Definition | Short name |
|--------------------|------------------|---|------------|
| 0b0 | 0b0 | Secure Group 0 | G0S |
| 0b0 | 0b1 | Non-secure Group 1 | G1NS |
| 0b1 | 0b0 | Secure Group 1 | G1S |
| 0b1 | 0b1 | Reserved, treated as Non-secure Group 1 | - |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_IGRPMODR0

When affinity routing is not enabled for the Security state of an interrupt in GICR_IGRPMODR0, the corresponding bit is RES0 and equivalent functionality is provided by [GICD_IGRPMODR<n>](#) with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by [GICD_IGRPMODR<n>](#).

When [GICD_CTLR.ARE_S](#) == 0 or [GICD_CTLR.DS](#) == 1, GICR_IGRPMODR0 is RES0. An implementation can make this register RAZ/WI in this case.

When [GICD_CTLR.DS](#)==0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICR_IGRPMODR0 can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|----------------------|----------|--------|----------------|
| GIC Redistributor | SGI_base | 0x0D00 | GICR_IGRPMODR0 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_IGRPMODR<n>E, Interrupt Group Modifier Registers, n = 1 - 2

The GICR_IGRPMODR<n>E characteristics are:

Purpose

When [GICD_CTLR.DS](#)==0, this register together with the GICR_IGROUPR<n>E registers, controls whether the corresponding interrupt is in:

- Secure Group 0.
- Non-secure Group 1.
- When System register access is enabled, Secure Group 1.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICR_IGRPMODR<n>E are RES0.

When [GICD_CTLR.DS](#)==0, this register is Secure.

A copy of this register is provided for each Redistributor.

Attributes

GICR_IGRPMODR<n>E is a 32-bit register.

Field descriptions

| | | | | | |
|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|
| 31 | 30 | 29 | 28 | 27 | |
| Group_modifier_bit31 | Group_modifier_bit30 | Group_modifier_bit29 | Group_modifier_bit28 | Group_modifier_bit27 | Group_modifier_bit26 |

Group_modifier_bit<x>, bit [x], for x = 31 to 0

Group modifier bit. In implementations where affinity routing is enabled for the Security state of an interrupt, the bit that corresponds to the interrupt is concatenated with the equivalent bit in [GICR_IGROUPR<n>E](#) to form a 2-bit field that defines an interrupt group:

| Group modifier bit | Group status bit | Definition | Short name |
|--------------------|------------------|---|------------|
| 0b0 | 0b0 | Secure Group 0 | G0S |
| 0b0 | 0b1 | Non-secure Group 1 | G1NS |
| 0b1 | 0b0 | Secure Group 1 | G1S |
| 0b1 | 0b1 | Reserved, treated as Non-secure Group 1 | - |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_IGRPMODR<n>E number, n, is given by $n = (m-1024) \text{ DIV } 32$.
- The offset of the required GICR_IGRPMODR<n>E is $(0x000 + (4*n))$.
- The bit number of the required group modifier bit in this register is $(m-1024) \text{ MOD } 32$.

Accessing GICR_IGRPMODR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_IGRPMODR<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)==0, the register is RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_IGRPMODR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|------------------------|-------------------|
| GIC Redistributor | SGI_base | 0x0D00 + (4 * n) | GICR_IGRPMODR<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_IIDR, Redistributor Implementer Identification Register

The GICR_IIDR characteristics are:

Purpose

Provides information about the implementer and revision of the Redistributor.

Configuration

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states, this register is Common.

Attributes

GICR_IIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|------|----|----|----|---------|----|----|----|----------|----|----|----|-------------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ProductID | | | | | | | | RES0 | | | | Variant | | | | Revision | | | | Implementer | | | | | | | | | | | |

ProductID, bits [31:24]

Product Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [23:20]

Reserved, RES0.

Variant, bits [19:16]

Variant number. Typically, this field is used to distinguish product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

Revision number. Typically, this field is used to distinguish minor revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 code of the company that implemented the Redistributor:

- Bits [11:8] are the JEP106 continuation code of the implementer. For an Arm implementation, this field is 0x4.
- Bit [7] is always 0.
- Bits [6:0] are the JEP106 identity code of the implementer. For an Arm implementation, bits [7:0] are therefore 0x3B.

Accessing GICR_IIDR

GICR_IIDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|---------|--------|-----------|
| GIC Redistributor | RD_base | 0x0004 | GICR_IIDR |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_INMIR0, Non-maskable Interrupt Register for PPIs.

The GICR_INMIR0 characteristics are:

Purpose

Controls whether the corresponding PPI has the non-maskable property.

Configuration

This register is present only when FEAT_GICv3_NMI is implemented. Otherwise, direct accesses to GICR_INMIR0 are RES0.

When [GICD_TYPER.NMI](#) is 0, this register is RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_INMIR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | |
|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 |
| nmi31 | nmi30 | nmi29 | nmi28 | nmi27 | nmi26 | nmi25 | nmi24 | nmi23 | nmi22 | nmi21 | nmi20 | nmi19 | nmi18 | nmi17 | nmi16 | nmi15 | nmi14 |

nmi<x>, bit [x], for x = 31 to 0

Non-maskable property.

| nmi<x> | Meaning |
|--------|--|
| 0b0 | Interrupt does not have the non-maskable property. |
| 0b1 | Interrupt has the non-maskable property. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

If affinity routing is disabled for the Security state of an interrupt, the bit is RES0.

Accessing GICR_INMIR0

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_INMIR0 can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|--------|-------------|
| GIC Redistributor | SGI_base | 0x0F80 | GICR_INMIR0 |

Accesses on this interface are **RW**.

GICR_INMIR<n>E, Non-maskable Interrupt Registers for Extended PPIs, x = 1 to 2., n = 1 - 2

The GICR_INMIR<n>E characteristics are:

Purpose

Controls whether the corresponding Extended PPI has the non-maskable property.

Configuration

This register is present only when FEAT_GICv3p1 is implemented and FEAT_GICv3_NMI is implemented. Otherwise, direct accesses to GICR_INMIR<n>E are RES0.

When [GICR_TYPER](#).PPInum is 0b0000 or [GICD_TYPER](#).NMI is 0, these registers are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_INMIR<n>E is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | |
|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|-----------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 |
| nmi31 | nmi30 | nmi29 | nmi28 | nmi27 | nmi26 | nmi25 | nmi24 | nmi23 | nmi22 | nmi21 | nmi20 | nmi19 | nmi18 | nmi17 | nmi16 | nmi15 | nmi14 |

nmi<x>, bit [x], for x = 31 to 0

Non-maskable property.

| nmi<x> | Meaning |
|--------|--|
| 0b0 | Interrupt does not have the non-maskable property. |
| 0b1 | Interrupt has the non-maskable property. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

If affinity routing is disabled for the Security state of an interrupt, the bit is RES0.

Accessing GICR_INMIR<n>E

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_INMIR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|------------------|----------------|
| GIC Redistributor | SGI_base | 0x0F80 + (4 * n) | GICR_INMIR<n>E |

Accesses on this interface are **RW**.

GICR_INVALLR, Redistributor Invalidate All Register

The GICR_INVALLR characteristics are:

Purpose

Invalidates any cached configuration data of all physical LPIs, causing the GIC to reload the interrupt configuration from the physical LPI Configuration table at the address specified by [GICR_PROPBASER](#).

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_INVALLR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| V | RES0 | | | | | | | | | | | | | | | vPEID | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

V, bit [63]
When FEAT_GICv4p1 is implemented:

Indicates whether the INTID is virtual or physical.

| V | Meaning |
|-----|-------------------------------------|
| 0b0 | Invalidate is for a physical INTID. |
| 0b1 | Invalidate is for a virtual INTID. |

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

vPEID, bits [47:32]
When FEAT_GICv4p1 is implemented:

When GICR_INVLPIR.V == 0, this field is RES0

When GICR_INVLPIR.V == 1, this field is the target vPEID of the invalidate.

Note

The size of this field is IMPLEMENTATION DEFINED, and is specified by the [GICD_TYPER2.VIL](#) and [GICD_TYPER2.VID](#) fields. Unimplemented bits are RES0.

Otherwise:

Reserved, RES0.

Bits [31:0]

Reserved, RES0.

Note

If any LPI has been forwarded to the PE and a valid write to GICR_INVALLR is received, the Redistributor must ensure it reloads its properties from memory. This has no effect on the forwarded LPI if it has already been activated.

Accessing GICR_INVALLR

This register is mandatory when any of the following are true:

- [GICR_TYPER](#).Direct is 1.
- [GICR_CTLR](#).IR is 1.
- GICv4.1 is implemented.

Otherwise, the functionality is IMPLEMENTATION DEFINED.

Writes to this register have no effect if no physical LPIs are currently stored in the local Redistributor cache.

GICR_INVALLR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|---------|--------|--------------|
| GIC Redistributor | RD_base | 0x00B0 | GICR_INVALLR |

Accesses on this interface are **WO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_INVLPIR, Redistributor Invalidate LPI Register

The GICR_INVLPIR characteristics are:

Purpose

Invalidates the cached configuration data of a specified LPI, causing the GIC to reload the interrupt configuration from the physical LPI Configuration table at the address specified by [GICR_PROPBASER](#).

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_INVLPIR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| V | RES0 | | | | | | | | | | | | | | | vPEID | | | | | | | | | | | | | | | |
| INTID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

V, bit [63]
When FEAT_GICv4p1 is implemented:

Indicates whether the INTID is virtual or physical.

| V | Meaning |
|-----|-------------------------------------|
| 0b0 | Invalidate is for a physical INTID. |
| 0b1 | Invalidate is for a virtual INTID. |

Otherwise:

Reserved, RES0.

Bits [62:48]

Reserved, RES0.

vPEID, bits [47:32]
When FEAT_GICv4p1 is implemented:

When GICR_INVLPIR.V == 0, this field is RES0

When GICR_INVLPIR.V == 1, this field is the target vPEID of the invalidate.

Note

The size of this field is IMPLEMENTATION DEFINED, and is specified by the [GICD_TYPER2.VIL](#) and [GICD_TYPER2.VID](#) fields. Unimplemented bits are RES0.

Otherwise:

Reserved, RES0.

INTID, bits [31:0]

The INTID of the physical LPI to be cleaned.

Note

The size of this field is IMPLEMENTATION DEFINED, and is specified by the [GICD_TYPER](#).IDbits field. Unimplemented bits are RES0.

Note

If any LPI has been forwarded to the PE and a valid write to GICR_INVLPIR is received, the Redistributor must ensure it reloads its properties from memory and apply any changes by retrieving and reforwarding the LPI as required. This has no effect on the forwarded LPI if it has already been activated.

Accessing GICR_INVLPIR

When written with a 32-bit write the data is zero-extended to 64 bits.

This register is mandatory when any of the following are true:

- [GICR_TYPER](#).Direct is 1.
- [GICR_CTLR](#).IR is 1.
- GICv4.1 is implemented.

Otherwise, the functionality is IMPLEMENTATION DEFINED.

Writes to this register have no effect if either:

- The specified LPI is not currently stored in the local Redistributor.
- The pINTID field corresponds to an unimplemented LPI.

GICR_INVLPIR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|---------|--------|--------------|
| GIC Redistributor | RD_base | 0x00A0 | GICR_INVLPIR |

Accesses on this interface are **WO**.

GICR_IPRIORITYR<n>, Interrupt Priority Registers, n = 0 - 7

The GICR_IPRIORITYR<n> characteristics are:

Purpose

Holds the priority of the corresponding interrupt for each SGI and PPI supported by the GIC.

Configuration

A copy of these registers is provided for each Redistributor.

These registers are configured as follows:

- GICR_IPRIORITYR0-GICR_IPRIORITYR3 store the priority of SGIs.
- GICR_IPRIORITYR4-GICR_IPRIORITYR7 store the priority of PPIs.

Attributes

GICR_IPRIORITYR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------|----|----|----|----|----|----|----|--------------------|----|----|----|----|----|----|----|--------------------|----|----|----|----|----|---|---|--------------------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Priority_offset_3B | | | | | | | | Priority_offset_2B | | | | | | | | Priority_offset_1B | | | | | | | | Priority_offset_0B | | | | | | | |

Priority_offset_3B, bits [31:24]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 3. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_2B, bits [23:16]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 2. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_1B, bits [15:8]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 1. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_0B, bits [7:0]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 0. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_IPRIORITYR<n>

These registers are used when affinity routing is enabled for the Security state of the interrupt. When affinity routing is not enabled the bits corresponding to the interrupt are RAZ/WI and [GICD_IPRIORITYR<n>](#) provides equivalent functionality.

These registers are used for SGIs and PPIs only. Equivalent functionality for SPIs is provided by [GICD_IPRIORITYR<n>](#).

These registers are byte-accessible.

When [GICD_CTLR](#).DS == 0:

- A field that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.
- A Non-secure access to a field that corresponds to a Non-secure Group 1 interrupt behaves as described in 'Software accesses of interrupt priority' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than one time. The effect of the change must be visible in finite time.

GICR_IPRIORITYR<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|------------------|--------------------|
| GIC Redistributor | SGI_base | 0x0400 + (4 * n) | GICR_IPRIORITYR<n> |

Accesses on this interface are **RW**.

GICR_IPRIORITYR<n>E, Interrupt Priority Registers (extended PPI range), n = 8 - 23

The GICR_IPRIORITYR<n>E characteristics are:

Purpose

Holds the priority of the corresponding interrupt for each extended PPI supported by the GIC.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICR_IPRIORITYR<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_IPRIORITYR<n>E is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------|----|----|----|----|----|----|----|--------------------|----|----|----|----|----|----|----|--------------------|----|----|----|----|----|---|---|--------------------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Priority_offset_3B | | | | | | | | Priority_offset_2B | | | | | | | | Priority_offset_1B | | | | | | | | Priority_offset_0B | | | | | | | |

Priority_offset_3B, bits [31:24]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 3. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_2B, bits [23:16]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 2. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_1B, bits [15:8]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 1. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Priority_offset_0B, bits [7:0]

Interrupt priority value from an IMPLEMENTATION DEFINED range, at byte offset 0. Lower priority values correspond to greater priority of the interrupt.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

For interrupt ID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_IPRIORITYR<n> number, n, is given by $n = (m-1024) \text{ DIV } 4$.
- The offset of the required GICR_IPRIORITYR<n>E register is $(0x400 + (4*n))$.
- The byte offset of the required Priority field in this register is $m \text{ MOD } 4$, where:
 - Byte offset 0 refers to register bits [7:0].
 - Byte offset 1 refers to register bits [15:8].
 - Byte offset 2 refers to register bits [23:16].
 - Byte offset 3 refers to register bits [31:24].

Accessing GICR_IPRIORITYR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISACTIVER<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)==0:

- A field that corresponds to a Group 0 or Secure Group 1 interrupt is RAZ/WI to Non-secure accesses.
- A Non-secure access to a field that corresponds to a Non-secure Group 1 interrupt behaves as described in Software accesses of interrupt priority.

Bits corresponding to unimplemented interrupts are RAZ/WI.

Note

Implementations must ensure that an interrupt that is pending at the time of the write uses either the old value or the new value and must ensure that the interrupt is neither lost nor handled more than once. The effect of the change must be visible in finite time.

GICR_IPRIORITYR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|------------------|---------------------|
| GIC Redistributor | SGI_base | 0x0400 + (4 * n) | GICR_IPRIORITYR<n>E |

Accesses on this interface are **RW**.

GICR_ISACTIVER0, Interrupt Set-Active Register 0

The GICR_ISACTIVER0 characteristics are:

Purpose

Activates the corresponding SGI or PPI. These registers are used when saving and restoring GIC state.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ISACTIVER0 is a 32-bit register.

Field descriptions

| | | | | | | |
|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 |
| Set_active_bit31 | Set_active_bit30 | Set_active_bit29 | Set_active_bit28 | Set_active_bit27 | Set_active_bit26 | Set_active_bit25 |

Set_active_bit<x>, bit [x], for x = 31 to 0

Adds the active state to interrupt number x. Reads and writes have the following behavior:

| Set_active_bit<x> | Meaning |
|-------------------|---|
| 0b0 | If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is active, or is active and pending. If written, activates the corresponding interrupt, if the interrupt is not already active. If the interrupt is already active, the write has no effect. After a write of 1 to this bit, a subsequent read of this bit returns 1. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_ISACTIVER0

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISACTIVER0, the corresponding bit is RAZ/WI and equivalent functionality is provided by [GICD_ISACTIVER<n>](#) with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by [GICD_ISACTIVER<n>](#).

When [GICD_CTLR](#).DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

GICR_ISACTIVER0 can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|-------|--------|----------|
|-----------|-------|--------|----------|

GICR_ISACTIVER0, Interrupt Set-Active Register 0

| | | | |
|----------------------|----------|--------|-----------------|
| GIC Redistributor | SGI_base | 0x0300 | GICR_ISACTIVER0 |
|----------------------|----------|--------|-----------------|

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ISACTIVER<n>E, Interrupt Set-Active Registers, n = 1 - 2

The GICR_ISACTIVER<n>E characteristics are:

Purpose

Adds the active state to the corresponding PPI.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICR_ISACTIVER<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ISACTIVER<n>E is a 32-bit register.

Field descriptions

| | | | | | | |
|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 |
| Set_active_bit31 | Set_active_bit30 | Set_active_bit29 | Set_active_bit28 | Set_active_bit27 | Set_active_bit26 | Set_active_bit25 |

Set_active_bit<x>, bit [x], for x = 31 to 0

For the extended PPIs, adds the active state to interrupt number x. Reads and writes have the following behavior:

| Set_active_bit<x> | Meaning |
|-------------------|---|
| 0b0 | If read, indicates that the corresponding interrupt is not active, and is not active and pending. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is active, or active and pending on this PE. If written, activates the corresponding interrupt, if the interrupt is not already active. If the interrupt is already active, the write has no effect. After a write of 1 to this bit, a subsequent read of this bit returns 1. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ISACTIVER<n>E number, n, is given by $n = (m-1024) \text{ DIV } 32$.
- The offset of the required GICR_ISACTIVER<n>E is $(0x200 + (4*n))$.
- The bit number of the required group modifier bit in this register is $(m-1024) \text{ MOD } 32$.

Accessing GICR_ISACTIVER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISACTIVER<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)=0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_ISACTIVER<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|------------------|--------------------|
| GIC Redistributor | SGI_base | 0x0300 + (4 * n) | GICR_ISACTIVER<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ISENABLER0, Interrupt Set-Enable Register 0

The GICR_ISENABLER0 characteristics are:

Purpose

Enables forwarding of the corresponding SGI or PPI to the CPU interfaces.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ISENABLER0 is a 32-bit register.

Field descriptions

| | | | | | | |
|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 |
| Set_enable_bit31 | Set_enable_bit30 | Set_enable_bit29 | Set_enable_bit28 | Set_enable_bit27 | Set_enable_bit26 | Set_enable_bit25 |

Set_enable_bit<x>, bit [x], for x = 31 to 0

For PPIs and SGIs, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

| Set_enable_bit<x> | Meaning |
|-------------------|---|
| 0b0 | If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect. |
| 0b1 | If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 1. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Accessing GICR_ISENABLER0

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISENABLER0, the corresponding bit is RAZ/WI and equivalent functionality is provided by [GICD_ISENABLER<n>](#) with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by [GICD_ISENABLER<n>](#).

When [GICD_CTLR](#).DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

GICR_ISENABLER0 can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|--------|-----------------|
| GIC Redistributor | SGI_base | 0x0100 | GICR_ISENABLER0 |

Accesses on this interface are **RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ISENABLER<n>E, Interrupt Set-Enable Registers, n = 1 - 2

The GICR_ISENABLER<n>E characteristics are:

Purpose

Enables forwarding of the corresponding PPI to the CPU interfaces.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICR_ISENABLER<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ISENABLER<n>E is a 32-bit register.

Field descriptions

| | | | | | | |
|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|----------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 |
| Set_enable_bit31 | Set_enable_bit30 | Set_enable_bit29 | Set_enable_bit28 | Set_enable_bit27 | Set_enable_bit26 | Set_enable_bit25 |

Set_enable_bit<x>, bit [x], for x = 31 to 0

For the extended PPI range, controls the forwarding of interrupt number x to the CPU interface. Reads and writes have the following behavior:

| Set_enable_bit<x> | Meaning |
|-------------------|---|
| 0b0 | If read, indicates that forwarding of the corresponding interrupt is disabled. If written, has no effect. |
| 0b1 | If read, indicates that forwarding of the corresponding interrupt is enabled. If written, enables forwarding of the corresponding interrupt. After a write of 1 to this bit, a subsequent read of this bit returns 1. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ISENABLER<n>E number, n, is given by $n = (m-1024) \text{ DIV } 32$.
- The offset of the required GICR_ISENABLER<n>E is $(0 \times 100 + (4 \times n))$.
- The bit number of the required group modifier bit in this register is $(m-1024) \text{ MOD } 32$.

Accessing GICR_ISENABLER<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISENABLER<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)=0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_ISENABLER<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|----------------------|----------|------------------------|--------------------|
| GIC Redistributor | SGI_base | 0x0100 + (4 * n) | GICR_ISENABLER<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ISPENDR0, Interrupt Set-Pending Register 0

The GICR_ISPENDR0 characteristics are:

Purpose

Adds the pending state to the corresponding SGI or PPI.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_ISPENDR0 is a 32-bit register.

Field descriptions

| | | | | | | |
|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|-----------------------------------|--------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | Se |
| Set_pending_bit31 | Set_pending_bit30 | Set_pending_bit29 | Set_pending_bit28 | Set_pending_bit27 | Set_pending_bit26 | Se |

Set_pending_bit<x>, bit [x], for x = 31 to 0

For PPIs and SGIs, adds the pending state to interrupt number x. Reads and writes have the following behavior:

| Set_pending_bit<x> | Meaning |
|--------------------|---|
| 0b0 | If read, indicates that the corresponding interrupt is not pending on this PE. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is pending, or active and pending on this PE. If written, changes the state of the corresponding interrupt from inactive to pending, or from active to active and pending. This has no effect in the following cases: <ul style="list-style-type: none">If the interrupt is already pending because of a write to GICR_ISPENDR0.If the interrupt is already pending because the corresponding interrupt signal is asserted. In this case, the interrupt remains pending if the interrupt signal is deasserted. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_ISPENDR0

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISPENDR0, the corresponding bit is RAZ/WI and equivalent functionality is provided by [GICD_ISPENDR<n>](#) with n=0.

This register only applies to SGIs (bits [15:0]) and PPIs (bits [31:16]). For SPIs, this functionality is provided by [GICD_ISPENDR<n>](#).

When [GICD_CTLR](#).DS == 0, bits corresponding to Secure SGIs and PPIs are RAZ/WI to Non-secure accesses.

GICR_ISPENDR0 can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|----------------------|----------|--------|---------------|
| GIC Redistributor | SGI_base | 0x0200 | GICR_ISPENDR0 |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_ISPENDR<n>E, Interrupt Set-Pending Registers, n = 1 - 2

The GICR_ISPENDR<n>E characteristics are:

Purpose

Adds the pending state to the corresponding PPI.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICR_ISPENDR<n>E are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_ISPENDR<n>E is a 32-bit register.

Field descriptions

| | | | | | | |
|-------------------|-------------------|-------------------|-------------------|-------------------|-------------------|----|
| 31 | 30 | 29 | 28 | 27 | 26 | Se |
| Set_pending_bit31 | Set_pending_bit30 | Set_pending_bit29 | Set_pending_bit28 | Set_pending_bit27 | Set_pending_bit26 | Se |

Set_pending_bit<x>, bit [x], for x = 31 to 0

For the extended PPIs, adds the pending state to interrupt number x. Reads and writes have the following behavior:

| Set_pending_bit<x> | Meaning |
|--------------------|--|
| 0b0 | If read, indicates that the corresponding interrupt is not pending on this PE. If written, has no effect. |
| 0b1 | If read, indicates that the corresponding interrupt is pending, or active and pending on this PE. If written, changes the state of the corresponding interrupt from inactive to pending, or from active to active and pending. This has no effect in the following cases: <ul style="list-style-type: none">• If the interrupt is already pending because of a write to GICR_ISPENDR<n>E.• If the interrupt is already pending because the corresponding interrupt signal is asserted. In this case, the interrupt remains pending if the interrupt signal is deasserted. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

For INTID m, when DIV and MOD are the integer division and modulo operations:

- The corresponding GICR_ISPENDR<n>E number, n, is given by $n = (m-1024) \text{ DIV } 32$.
- The offset of the required GICR_ISPENDR<n>E is $(0 \times 200 + (4 \times n))$.
- The bit number of the required group modifier bit in this register is $(m-1024) \text{ MOD } 32$.

Accessing GICR_ISPENDR<n>E

When affinity routing is not enabled for the Security state of an interrupt in GICR_ISPENDR<n>E, the corresponding bit is RES0.

When [GICD_CTLR.DS](#)==0, bits corresponding to Secure PPIs are RAZ/WI to Non-secure accesses.

Bits corresponding to unimplemented interrupts are RAZ/WI.

GICR_ISPENDR<n>E can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|----------|------------------------|------------------|
| GIC Redistributor | SIG_base | 0x0200 + (4 * n) | GICR_ISPENDR<n>E |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_MPAMIDR, Report maximum PARTID and PMG Register

The GICR_MPAMIDR characteristics are:

Purpose

Reports the maximum support PARTID and PMG values.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICR_MPAMIDR are RES0.

A copy of this register is provided for each Redistributor.

When [GICR_TYPER](#).MPAM==0, this register is RES0.

Attributes

GICR_MPAMIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|-----------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | PMGmax | | | | | | | | PARTIDmax | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

PMGmax, bits [23:16]

Maximum PMG value supported.

PARTIDmax, bits [15:0]

Maximum PARTID value supported.

Accessing GICR_MPAMIDR

GICR_MPAMIDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|---------|--------|--------------|
| GIC Redistributor | RD_base | 0x0018 | GICR_MPAMIDR |

Accesses on this interface are **RO**.

GICR_NSACR, Non-secure Access Control Register

The GICR_NSACR characteristics are:

Purpose

Enables Secure software to permit Non-secure software to create SGIs targeting the PE connected to this Redistributor by writing to [ICC_SGI1R_EL1](#), [ICC_ASGI1R_EL1](#) or [ICC_SGI0R_EL1](#).

For more information, see 'Forwarding an SGI to a target PE' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Configuration

For a description on when a write to [ICC_SGI0R_EL1](#), [ICC_SGI1R_EL1](#) or [ICC_ASGI1R_EL1](#) is permitted to generate an interrupt, see 'Use of control registers for SGI forwarding' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Attributes

GICR_NSACR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | |
|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|----------------------------|----------------------------|----------------------------|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 |
| NS_access15 | NS_access14 | NS_access13 | NS_access12 | NS_access11 | NS_access10 | NS_access9 | NS_access8 | NS_access7 | | | | | | | | | |

NS_access<x>, bits [2x+1:2x], for x = 15 to 0

Configures the level of Non-secure access permitted when the SGI is in Secure Group 0 or Secure Group 1, as defined from [GICR_IGROUPRO](#) and [GICR_IGRPMODR0](#). A field is provided for each SGI. The possible values of each 2-bit field are:

| NS_access<x> | Meaning |
|--------------|---|
| 0b00 | Non-secure writes are not permitted to generate Secure Group 0 SGIs or Secure Group 1 SGIs. |
| 0b01 | Non-secure writes are permitted to generate a Secure Group 0 SGI. |
| 0b10 | As 0b01, but additionally Non-secure writes to are permitted to generate a Secure Group 1 SGI. |
| 0b11 | Reserved. If the field is programmed to the reserved value, then the hardware will treat the field as if it has been programmed to an IMPLEMENTATION DEFINED choice of the valid values. However, to maintain the principle that as the value increases additional accesses are permitted Arm strongly recommends that implementations treat this value as 0b10. It is IMPLEMENTATION DEFINED whether the value read back is the value programmed or the valid value chosen. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_NSACR

This register is used when affinity routing is enabled. When affinity routing is not enabled for the Security state of the interrupt, [GICD_NSACR<n>](#) with n=0 provides equivalent functionality.

This register does not support PPIs.

GICR_NSACR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|----------------------|----------|--------|------------|
| GIC Redistributor | SGI_base | 0x0E00 | GICR_NSACR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 1 accesses to this register are **RAZ/WI**.
- When GICD_CTLR.DS == 0 and an access is Secure accesses to this register are **RW**.
- When GICD_CTLR.DS == 0 and an access is Non-secure accesses to this register are **RAZ/WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRoot() accesses to this register are **RW**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRealm() accesses to this register are **RAZ/WI**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_PARTIDR, Set PARTID and PMG Register

The GICR_PARTIDR characteristics are:

Purpose

Sets the PARTID and PMG values used for memory accesses by the Redistributor.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GICR_PARTIDR are RES0.

A copy of this register is provided for each Redistributor.

When [GICR_TYPER](#).MPAM==0, this register is RES0.

Attributes

GICR_PARTIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | PMG | | | | | | | | PARTID | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

PMG value used when Redistributor accesses memory.

It is IMPLEMENTATION DEFINED whether bits not needed to represent PMG values in the range 0 to PMG_MAX are stateful or RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

PARTID, bits [15:0]

PARTID value used when Redistributor accesses memory.

It is IMPLEMENTATION DEFINED whether bits not needed to represent PARTID values in the range 0 to PARTID_MAX are stateful or RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Accessing GICR_PARTIDR

GICR_PARTIDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|----------------------|---------|--------|--------------|
| GIC Redistributor | RD_base | 0x001C | GICR_PARTIDR |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_PENDBASER, Redistributor LPI Pending Table Base Address Register

The GICR_PENDBASER characteristics are:

Purpose

Specifies the base address of the LPI Pending table, and the Shareability and Cacheability of accesses to the LPI Pending table.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_PENDBASER is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------|-----|------|------------|----|----|----|------|----|----|----|------------------|----|----|----|----|------|----|--------------|----|------------|----|------|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| RES0 | PTZ | RES0 | OuterCache | | | | RES0 | | | | Physical_Address | | | | | | | | | | | | | | | | | | | | | |
| Physical_Address | | | | | | | | | | | | | | | | RES0 | | Shareability | | InnerCache | | RES0 | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Bit [63]

Reserved, RES0.

PTZ, bit [62]

Pending Table Zero. Indicates to the Redistributor whether the LPI Pending table is zero when [GICR_CTLR.EnableLPIs](#) == 1.

This field is WO, and reads as 0.

| PTZ | Meaning |
|-----|---|
| 0b0 | The LPI Pending table is not zero, and contains live data. |
| 0b1 | The LPI Pending table is zero. Software must ensure the LPI Pending table is zero before this value is written. |

Bits [61:59]

Reserved, RES0.

OuterCache, bits [58:56]

Indicates the Outer Cacheability attributes of accesses to the LPI Pending table.

| OuterCache | Meaning |
|------------|---|
| 0b000 | Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability. |
| 0b001 | Normal Outer Non-cacheable. |
| 0b010 | Normal Outer Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Outer Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Outer Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Outer Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-back. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52]

Reserved, RES0.

Physical_Address, bits [51:16]

Bits [51:16] of the physical address containing the LPI Pending table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [15:12]

Reserved, RES0.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the LPI Pending table.

| Shareability | Meaning |
|--------------|----------------------------|
| 0b00 | Non-shareable. |
| 0b01 | Inner Shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Reserved. Treated as 0b00. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

InnerCache, bits [9:7]

Indicates the Inner Cacheability attributes of accesses to the LPI Pending table.

| InnerCache | Meaning |
|------------|--|
| 0b000 | Device-nGnRnE. |
| 0b001 | Normal Inner Non-cacheable. |
| 0b010 | Normal Inner Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Inner Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Inner Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Inner Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-back. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [6:0]

Reserved, RES0.

Accessing GICR_PENDBASER

Having the GICR_PENDBASER OuterCache, Shareability or InnerCache fields programmed to different values on different Redistributors with [GICR_CTLR.EnableLPIs == 1](#) in the system is UNPREDICTABLE.

Changing GICR_PENDBASER with [GICR_CTLR.EnableLPIs == 1](#) is UNPREDICTABLE.

GICR_PENDBASER can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|---------|--------|----------------|
| GIC Redistributor | RD_base | 0x0078 | GICR_PENDBASER |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_PROPBASER, Redistributor Properties Base Address Register

The GICR_PROPBASER characteristics are:

Purpose

Specifies the base address of the LPI Configuration table, and the Shareability and Cacheability of accesses to the LPI Configuration table.

Configuration

A copy of this register is provided for each Redistributor.

An implementation might make this register RO, for example to correspond to an LPI Configuration table in read-only memory.

Attributes

GICR_PROPBASER is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------|----|----|----|------------|----|----|----|------|----|----|----|------------------|----|----|----|--------------|----|----|----|------------|----|----|----|------|----|----|----|--------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | OuterCache | | | | RES0 | | | | Physical_Address | | | | | | | | | | | | | | | | | | | |
| Physical_Address | | | | | | | | | | | | | | | | Shareability | | | | InnerCache | | | | RES0 | | | | IDbits | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:59]

Reserved, RES0.

OuterCache, bits [58:56]

Indicates the Outer Cacheability attributes of accesses to the LPI Configuration table.

| OuterCache | Meaning |
|------------|---|
| 0b000 | Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability. |
| 0b001 | Normal Outer Non-cacheable. |
| 0b010 | Normal Outer Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Outer Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Outer Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Outer Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-back. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52]

Reserved, RES0.

Physical_Address, bits [51:12]

Bits [51:12] of the physical address containing the LPI Configuration table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the LPI Configuration table.

| Shareability | Meaning |
|--------------|----------------------------|
| 0b00 | Non-shareable. |
| 0b01 | Inner Shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Reserved. Treated as 0b00. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

InnerCache, bits [9:7]

Indicates the Inner Cacheability attributes of accesses to the LPI Configuration table.

| InnerCache | Meaning |
|------------|--|
| 0b000 | Device-nGnRnE. |
| 0b001 | Normal Inner Non-cacheable. |
| 0b010 | Normal Inner Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Inner Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Inner Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Inner Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-back. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IDbits, bits [4:0]

The number of bits of LPI INTID supported, minus one, by the LPI Configuration table starting at Physical_Address.

If the value of this field is larger than the value of [GICD_TYPER.IDbits](#), the [GICD_TYPER.IDbits](#) value applies.

If the value of this field is less than 0b1101, indicating that the largest INTID is less than 8192 (the smallest LPI interrupt ID), the GIC will behave as if all physical LPis are out of range.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GICR_PROPBASER

It is IMPLEMENTATION DEFINED whether GICR_PROPBASER can be set to different values on different Redistributors. [GICR_TYPER.CommonLPIAff](#) identifies the Redistributors that must have GICR_PROPBASER set to the same values whenever [GICR_CTLR.EnableLPIs](#) == 1.

Setting different values in different copies of GICR_PROPBASER on Redistributors that are required to use a common LPI Configuration table when [GICR_CTLR.EnableLPIs](#) == 1 leads to UNPREDICTABLE behavior.

Other restrictions apply when a Redistributor caches information from GICR_PROPBASER. For more information, see 'LPI Configuration tables' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

GICR_PROPBASER can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|---------|--------|----------------|
| GIC Redistributor | RD_base | 0x0070 | GICR_PROPBASER |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_SETLPIR, Set LPI Pending Register

The GICR_SETLPIR characteristics are:

Purpose

Generates an LPI by setting the pending state of the specified LPI.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_SETLPIR is a 64-bit register.

Field descriptions

When GICR_TYPER.DirectLPI == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | pINTID | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:32]

Reserved, RES0.

pINTID, bits [31:0]

The INTID of the physical LPI to be generated.

Note

The size of this field is IMPLEMENTATION DEFINED, and is specified by the [GICD_TYPER.IDbits](#) field. Unimplemented bits are RES0.

When GICR_TYPER.DirectLPI == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

IMPLEMENTATION DEFINED, bits [63:0]

IMPLEMENTATION DEFINED.

Accessing GICR_SETLPIR

When written with a 32-bit write the data is zero-extended to 64 bits.

This register is mandatory in an implementation that supports LPis and does not include an ITS. The functionality is IMPLEMENTATION DEFINED in an implementation that does include an ITS.

Writes to this register have no effect if either:

- The pINTID field corresponds to an LPI that is already pending.
- The pINTID field corresponds to an unimplemented LPI.
- [GICR_CTLR.EnableLPis == 0](#).

GICR_SETLPIR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|----------------------|---------|--------|--------------|
| GIC Redistributor | RD_base | 0x0040 | GICR_SETLPIR |

Accesses on this interface are **WO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_STATUSR, Error Reporting Status Register

The GICR_STATUSR characteristics are:

Purpose

Provides software with a mechanism to detect:

- Accesses to reserved locations.
- Writes to read-only locations.
- Reads of write-only locations.

Configuration

A copy of this register is provided for each Redistributor.

If the GIC implementation supports two Security states this register is Banked to provide Secure and Non-secure copies.

Attributes

GICR_STATUSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|------|----|-----|---|-----|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | WROD | | | | RWOD | | WRD | | RRD | | | | | | | |

Bits [31:4]

Reserved, RES0.

WROD, bit [3]

Write to an RO location.

| WROD | Meaning |
|------|--|
| 0b0 | Normal operation. |
| 0b1 | A write to an RO location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

RWOD, bit [2]

Read of a WO location.

| RWOD | Meaning |
|------|--|
| 0b0 | Normal operation. |
| 0b1 | A read of a WO location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

WRD, bit [1]

Write to a reserved location.

| WRD | Meaning |
|-----|---|
| 0b0 | Normal operation. |
| 0b1 | A write to a reserved location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

RRD, bit [0]

Read of a reserved location.

| RRD | Meaning |
|-----|--|
| 0b0 | Normal operation. |
| 0b1 | A read of a reserved location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

Accessing GICR_STATUSR

This is an optional register. If the register is not implemented, the location is RAZ/WI.

GICR_STATUSR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|---------|--------|------------------|
| GIC Redistributor | RD_base | 0x0010 | GICR_STATUSR (S) |

This interface is accessible as follows:

- When an access is Secure accesses to this register are **RW**.
- When FEAT_RME is implemented and IsAccessRoot() accesses to this register are **RW**.

| Component | Frame | Offset | Instance |
|-------------------|---------|--------|-------------------|
| GIC Redistributor | RD_base | 0x0010 | GICR_STATUSR (NS) |

This interface is accessible as follows:

- When an access is Non-secure accesses to this register are **RW**.
- When FEAT_RME is implemented and IsAccessRealm() accesses to this register are **RW**.

GICR_SYNCR, Redistributor Synchronize Register

The GICR_SYNCR characteristics are:

Purpose

Indicates completion of register based invalidate operations.

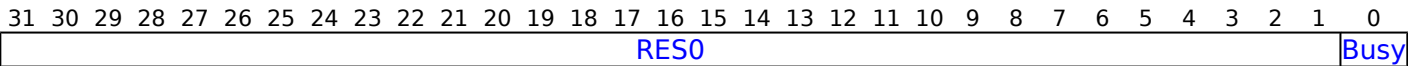
Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_SYNCR is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

Busy, bit [0]

Indicates completion of invalidation operations

| Busy | Meaning |
|------|--|
| 0b0 | No operations are in progress. |
| 0b1 | A write is in progress to one or more of the following registers: <ul style="list-style-type: none">GICR_INVLPIR.GICR_INVALLR.GICv3, GICR_CLRLPIR. |

This field tracks operations initiated on the same Redistributor.

Accessing GICR_SYNCR

When this register is accessed, it is optional that an implementation might wait until all operations are complete before returning a value, in which case GICR_SYNCR.Busy is always 0.

This register is mandatory when any of the following are true:

- GICR_TYPER.Direct is 1.
- GICR_CTLR.IR is 1.
- GICv4.1 is implemented.

Otherwise, the functionality is IMPLEMENTATION DEFINED.

GICR_SYNCR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|-------|--------|----------|
|-----------|-------|--------|----------|

GICR_SYNCR, Redistributor Synchronize Register

| | | | |
|----------------------|---------|--------|------------|
| GIC Redistributor | RD_base | 0x00C0 | GICR_SYNCR |
|----------------------|---------|--------|------------|

Accesses on this interface are **RO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_TYPER, Redistributor Type Register

The GICR_TYPER characteristics are:

Purpose

Provides information about the configuration of this Redistributor.

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_TYPER is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------|-----|------|--------|-----|-----|------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|------|------|------|--------|-----|-------|-------|-------|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Affinity_Value | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PPI | num | VSGI | Common | LPI | Aff | Processor_Number | | | | | | | | | | | | | | | | RV | PEID | MPAM | DPGS | Last | Direct | LPI | Dirty | VLPIS | PLPIS |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Affinity_Value, bits [63:32]

The identity of the PE associated with this Redistributor.

Bits [63:56] provide Aff3, the Affinity level 3 value for the Redistributor.

Bits [55:48] provide Aff2, the Affinity level 2 value for the Redistributor.

Bits [47:40] provide Aff1, the Affinity level 1 value for the Redistributor.

Bits [39:32] provide Aff0, the Affinity level 0 value for the Redistributor.

PPInum, bits [31:27]

When FEAT_GICv3p1 is implemented:

The value derived from this field specifies the maximum PPI INTID that a GIC implementation can support. An implementation might not implement all PPIs up to this maximum.

| PPInum | Meaning |
|---------|----------------------------|
| 0b00000 | Maximum PPI INTID is 31. |
| 0b00001 | Maximum PPI INTID is 1087. |
| 0b00010 | Maximum PPI INTID is 1119. |

All other values are reserved.

Otherwise:

Reserved, RES0.

VSGI, bit [26]

When FEAT_GICv4p1 is implemented:

Indicates whether vSGIs are supported.

| VSGI | Meaning |
|------|---|
| 0b0 | Direct injection of SGIs not supported. |
| 0b1 | Direct injection of SGIs supported. |

Otherwise:

Reserved, RES0.

CommonLPIAff, bits [25:24]

The affinity level at which Redistributors share an LPI Configuration table.

| CommonLPIAff | Meaning |
|--------------|--|
| 0b00 | All Redistributors must share an LPI Configuration table. |
| 0b01 | All Redistributors with the same Aff3 value must share an LPI Configuration table. |
| 0b10 | All Redistributors with the same Aff3.Aff2 value must share an LPI Configuration table. |
| 0b11 | All Redistributors with the same Aff3.Aff2.Aff1 value must share an LPI Configuration table. |

Processor_Number, bits [23:8]

A unique identifier for the PE. When [GITS_TYPER](#).PTA == 0, an ITS uses this field to identify the interrupt target.

When affinity routing is disabled for a Security state, this field indicates which [GICD_ITARGETSR<n>](#) corresponds to this Redistributor.

RVPEID, bit [7]

When FEAT_GICv4p1 is implemented:

Indicates how the resident vPE is specified.

| RVPEID | Meaning |
|--------|---|
| 0b0 | GICR_VPENDBASER records the address of the vPE's Virtual Pending Table. |
| 0b1 | GICR_VPENDBASER records vPEID. |

Otherwise:

Reserved, RES0.

MPAM, bit [6]

When FEAT_GICv3p1 is implemented:

MPAM

| MPAM | Meaning |
|------|---------------------|
| 0b0 | MPAM not supported. |
| 0b1 | MPAM supported. |

Otherwise:

Reserved, RES0.

DPGS, bit [5]

Sets support for [GICR_CTLR](#).DPG* bits.

| DPGS | Meaning |
|------|--|
| 0b0 | GICR_CTLR.DPG* bits are not supported. |
| 0b1 | GICR_CTLR.DPG* bits are supported. |

Last, bit [4]

Indicates whether this Redistributor is the highest-numbered Redistributor in a series of contiguous Redistributor pages.

| Last | Meaning |
|------|---|
| 0b0 | This Redistributor is not the highest-numbered Redistributor in a series of contiguous Redistributor pages. |
| 0b1 | This Redistributor is the highest-numbered Redistributor in a series of contiguous Redistributor pages. |

DirectLPI, bit [3]

Indicates whether this Redistributor supports direct injection of LPIs.

| DirectLPI | Meaning |
|-----------|--|
| 0b0 | This Redistributor does not support direct injection of LPIs. The GICR_SETLPIR , GICR_CLRLPIR , GICR_INVLPPIR , GICR_INVALLR , and GICR_SYNCR registers are either not implemented, or have an IMPLEMENTATION DEFINED purpose. |
| 0b1 | This Redistributor supports direct injection of LPIs. The GICR_SETLPIR , GICR_CLRLPIR , GICR_INVLPPIR , GICR_INVALLR , and GICR_SYNCR registers are implemented. |

Dirty, bit [2]

Controls the functionality of [GICR_VPENDBASER](#). Dirty.

| Dirty | Meaning |
|-------|---|
| 0b0 | GICR_VPENDBASER .Dirty is UNKNOWN when GICR_VPENDBASER .Valid == 1. |
| 0b1 | GICR_VPENDBASER .Dirty indicates when the Virtual Pending Table has been parsed when GICR_VPENDBASER .Valid is written from 0 to 1. |

When [GICR_TYPER.VLPIS](#) == 0, this field is RES0.

Note

In GICv4p1 implementations this field is RES1.

VLPIS, bit [1]

Indicates whether the GIC implementation supports virtual LPIs and the direct injection of virtual LPIs.

| VLPIS | Meaning |
|-------|---|
| 0b0 | The implementation does not support virtual LPIs or the direct injection of virtual LPIs. |
| 0b1 | The implementation supports virtual LPIs and the direct injection of virtual LPIs. |

Note

In GICv3 implementations this field is RES0.

PLPIS, bit [0]

Indicates whether the GIC implementation supports physical LPis.

| PLPIS | Meaning |
|-------|--|
| 0b0 | The implementation does not support physical LPis. |
| 0b1 | The implementation supports physical LPis. |

Accessing GICR_TYPER

GICR_TYPER can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|---------|--------|------------|
| GIC Redistributor | RD_base | 0x0008 | GICR_TYPER |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_VPENDBASER, Virtual Redistributor LPI Pending Table Base Address Register

The GICR_VPENDBASER characteristics are:

Purpose

Specifies the base address of the memory that holds the virtual LPI Pending table for the currently scheduled virtual machine.

Configuration

Attributes

GICR_VPENDBASER is a 64-bit register.

Field descriptions

When FEAT_GICv4 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------|------|-------------|-------|------|------------|------|------------------|----|----|----|----|----|----|----|------|--------------|------------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Valid | IDAI | PendingLast | Dirty | RES0 | OuterCache | RES0 | Physical Address | | | | | | | | | | | | | | | | | | | | | | | | |
| Physical Address | | | | | | | | | | | | | | | RES0 | Shareability | InnerCache | RES0 | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Valid, bit [63]

This bit controls whether the virtual LPI Pending table is valid.

| Valid | Meaning |
|-------|--|
| 0b0 | The virtual LPI Pending table is not valid. No vPE is scheduled. |
| 0b1 | The virtual LPI Pending table is valid. A vPE is scheduled. |

Setting GICR_VPENDBASER.Valid == 1 when the associated CPU interface does not implement FEAT_GICv4 is UNPREDICTABLE.

Note

Software can determine whether a PE supports FEAT_GICv3 or FEAT_GICv4 by reading ID_AA64PFR0_EL1.

Writing a new value to any bit of GICR_VPENDBASER, other than GICR_VPENDBASER.Valid, when GICR_VPENDBASER.Valid==1 is UNPREDICTABLE.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

IDAI, bit [62]

Implementation Defined Area Invalid. Indicates whether the IMPLEMENTATION DEFINED area in the virtual LPI Pending table is valid.

| IDAI | Meaning |
|------|--|
| 0b0 | The IMPLEMENTATION DEFINED area is valid. |
| 0b1 | The IMPLEMENTATION DEFINED area is invalid and all pending interrupt information is held in the architecturally defined part of the virtual LPI Pending table. |

For more information, see 'LPI Pending tables' and 'Virtual LPI Configuration tables and virtual LPI Pending tables' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

PendingLast, bit [61]

Indicates whether there are pending and enabled interrupts for the last scheduled vPE.

This value is set by the implementation when GICR_VPENDBASER.Valid has been written from 1 to 0 and is otherwise UNKNOWN.

| PendingLast | Meaning |
|-------------|---|
| 0b0 | There are no pending and enabled interrupts for the last scheduled vPE. |
| 0b1 | There is at least one pending interrupt for the last scheduled vPE. It is IMPLEMENTATION DEFINED whether this bit is set when the only pending interrupts for the last scheduled vPE are not enabled. Arm deprecates setting PendingLast to 1 when the only pending interrupts for the last scheduled virtual machine are not enabled. |

When the GICR_VPENDBASER.Valid bit is written from 0 to 1, this bit is RES1.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Dirty, bit [60]

When GICR_VPENDBASER.Valid == 0:

Indicates whether a de-scheduling operation is in progress.

This field is read-only.

| Dirty | Meaning |
|-------|--|
| 0b0 | No de-scheduling operation in process. |
| 0b1 | De-scheduling operation in process. |

Writing 1 to GICR_VPENDBASER.Valid is UNPREDICTABLE while GICR_VPENDBASER.Dirty==1.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

When GICR_VPENDBASER.Valid == 1 and GICR_TYPER.Dirty == 1:

This field is read-only. Reports whether the Virtual Pending table has been parsed.

| Dirty | Meaning |
|-------|---|
| 0b0 | Parsing of the Virtual Pending Table has completed. |
| 0b1 | Parsing of the Virtual Pending Table has not completed. |

Writing 1 to GICR_VPENDBASER.Valid is UNPREDICTABLE while GICR_VPENDBASER.Dirty == 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Otherwise:

This field is read-only. This field is UNKNOWN.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Bit [59]

Reserved, RES0.

OuterCache, bits [58:56]

Indicates the Outer Cacheability attributes of accesses to virtual LPI Pending tables of vPEs targeting this Redistributor.

| OuterCache | Meaning |
|------------|---|
| 0b000 | Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability. |
| 0b001 | Normal Outer Non-cacheable. |
| 0b010 | Normal Outer Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Outer Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Outer Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Outer Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-back. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The Cacheability, Outer Cacheability and Shareability fields are used for accesses to the virtual LPI Pending table of resident and non-resident vPEs.

If the OuterCacheability attribute of the virtual LPI Pending tables that are associated with vPEs targeting the same Redistributor are different, behavior is UNPREDICTABLE.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52]

Reserved, RES0.

Physical_Address, bits [51:16]

Bits [51:16] of the physical address containing the virtual LPI Pending table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [15:12]

Reserved, RES0.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the virtual LPI Pending table.

| Shareability | Meaning |
|--------------|----------------------------|
| 0b00 | Non-shareable. |
| 0b01 | Inner Shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Reserved. Treated as 0b00. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The Cacheability, Outer Cacheability and Shareability fields are used for accesses to the virtual LPI Pending table of resident and non-resident vPEs.

If the Shareability attribute of the virtual LPI Pending tables that are associated with vPEs targeting the same Redistributor are different, behavior is UNPREDICTABLE.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

InnerCache, bits [9:7]

Indicates the Inner Cacheability attributes of accesses to the virtual LPI Pending table.

| InnerCache | Meaning |
|------------|--|
| 0b000 | Device-nGnRnE. |
| 0b001 | Normal Inner Non-cacheable. |
| 0b010 | Normal Inner Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Inner Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Inner Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Inner Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-back. |

The Cacheability, Outer Cacheability and Shareability fields are used for accesses to the virtual LPI Pending table of resident and non-resident vPEs.

If the InnerCacheability attribute of the virtual LPI Pending tables that are associated with vPEs targeting the same Redistributor are different, behavior is UNPREDICTABLE.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [6:0]

Reserved, RES0.

When FEAT_GICv4p1 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----------|-------------|-------|---------|---------|------|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Valid | Doorbell | PendingLast | Dirty | VGrp0En | VGrp1En | RES0 | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | vPEID | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Valid, bit [63]

This bit controls whether a vPE is scheduled:

| Valid | Meaning |
|-------|--|
| 0b0 | The virtual LPI Pending table is not valid. No vPE is scheduled. |
| 0b1 | The virtual LPI Pending table is valid. A vPE is scheduled. |

Setting GICR_VPENDBASER.Valid == 1 when the associated CPU interface does not implement FEAT_GICv4 is UNPREDICTABLE.

Note

Software can determine whether a PE supports FEAT_GICv3 or FEAT_GICv4 by reading ID_AA64PFR0_EL1.

Writing a new value to any bit of GICR_VPENDBASER, other than GICR_VPENDBASER.Valid, when GICR_VPENDBASER.Valid==1 is UNPREDICTABLE.

Setting GICR_VPENDBASER.Valid to 1 is UNPREDICTABLE if [GICR_VPROPBASER](#).Valid == 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Doorbell, bit [62]

When GICR_VPENDBASER.Valid is written from 1 to 0, this bit controls whether a default doorbell interrupt is requested for the descheduled vPE.

| Doorbell | Meaning |
|----------|--------------------------------|
| 0b0 | No default doorbell requested. |
| 0b1 | Default doorbell requested. |

When GICR_VPENDBASER.Valid is written from 1 to 0, if there are outstanding enabled pending interrupts then this bit is treated as 0.

When GICR_VPENDBASER.Valid is written from 1 to 0, if GICR_VPENDBASER.PendingLast is written as 1 then this bit is treated as 0.

When GICR_VPENDBASER.Valid == 1, reads return an UNKNOWN value.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

PendingLast, bit [61]

Indicates whether there are pending and enabled interrupts for the last scheduled vPE.

This value is set by the implementation when GICR_VPENDBASER.Valid is written from 1 to 0 and is otherwise UNKNOWN.

| PendingLast | Meaning |
|-------------|---|
| 0b0 | There are no pending and enabled interrupts for the last scheduled vPE. |
| 0b1 | There is at least one pending and enabled interrupt for the last scheduled vPE. |

When the GICR_VPENDBASER.Valid bit is written from 0 to 1, this bit is RES1.

When GICR_VPENDBASER.Valid is written from 1 to 0, if GICR_VPENDBASER.PendingLast is written as 1, then this bit is set to an UNKNOWN value.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Dirty, bit [60]**When GICR_VPENDBASER.Valid == 0:**

Read-only. Indicates whether a de-scheduling operation is in progress.

| Dirty | Meaning |
|-------|---|
| 0b0 | No de-scheduling operation in progress. |
| 0b1 | De-scheduling operation in progress. |

Writing 1 to GICR_VPENDBASER.Valid is UNPREDICTABLE while GICR_VPENDBASER.Dirty == 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Otherwise:

Read-only. Reports whether the Virtual Pending table has been parsed.

| Dirty | Meaning |
|-------|---|
| 0b0 | Parsing of the Virtual Pending Table is complete. |
| 0b1 | Parsing of the Virtual Pending Table has not completed. |

Writing 1 to GICR_VPENDBASER.Valid is UNPREDICTABLE while GICR_VPENDBASER.Dirty == 1.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

VGrp0En, bit [59]

Enable virtual Group 0 interrupts.

| VGrp0En | Meaning |
|---------|--|
| 0b0 | Forwarding of virtual Group 0 interrupts disabled. |
| 0b1 | Forwarding of virtual Group 0 interrupts enabled. |

Writing a new value to VGrp0En while [GICR_VPENDBASER.Valid==1](#) is CONSTRAINED UNPREDICTABLE:

- The update is ignored.
- The update is ignored for all purposes other than a direct read of the register.
- The virtual group enable is updated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

VGrp1En, bit [58]

Enable virtual Group 1 interrupts.

| VGrp1En | Meaning |
|---------|--|
| 0b0 | Forwarding of virtual Group 1 interrupts disabled. |
| 0b1 | Forwarding of virtual Group 1 interrupts enabled. |

Writing a new value to VGrp1En while [GICR_VPENDBASER.Valid==1](#) is CONSTRAINED UNPREDICTABLE:

- The update is ignored.
- The update is ignored for all purposes other than a direct read of the register.
- The virtual group enable is updated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Bits [57:16]

Reserved, RES0.

vPEID, bits [15:0]

When GICR_VPENDBASER.Valid == 1, ID of scheduled vPE.

When GICR_VPENDBASER.Valid == 1, if GICR_VPENDBASER.vPEID is set to a value greater than the configured vPEID width, the behavior of this field is CONSTRAINED UNPREDICTABLE:

- GICR_VPENDBASER.vPEID is treated as having an UNKNOWN valid value for all purposes other than a direct read of the register.
- GICR_VPENDBASER.Valid is treated as being set to 0 for all purposes other than a direct read of the register.

The size of this field is IMPLEMENTATION DEFINED, and is specified by the [GICD_TYPER2.VIL](#) and [GICD_TYPER2.VID](#) fields, unimplemented bits are RES0.

Accessing GICR_VPENDBASER

The effect of a write to this register is not guaranteed to be visible throughout the affinity hierarchy, as indicated by [GICR_CTLR.RWP](#) == 0.

GICR_VPENDBASER can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|-----------|--------|-----------------|
| GIC Redistributor | VLPI_base | 0x0078 | GICR_VPENDBASER |

Accesses on this interface are **RW**.

GICR_VPROPBASER, Virtual Redistributor Properties Base Address Register

The GICR_VPROPBASER characteristics are:

Purpose

Specifies the base address of the memory that holds the virtual LPI Configuration table for the currently scheduled virtual machine.

Configuration

This register is provided in FEAT_GICv4 implementations only.

Attributes

GICR_VPROPBASER is a 64-bit register.

Field descriptions

When FEAT_GICv4 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------|----|----|----|----|------------|----|----|------|----|----|----|------------------|----|----|----|----|----|----|----|--------------|----|------------|----|----|------|----|----|--------|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | OuterCache | | | RES0 | | | | Physical_Address | | | | | | | | | | | | | | | | | | | |
| Physical_Address | | | | | | | | | | | | | | | | | | | | Shareability | | InnerCache | | | RES0 | | | IDbits | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:59]

Reserved, RES0.

OuterCache, bits [58:56]

Indicates the Outer Cacheability attributes of accesses to the LPI Configuration table.

| OuterCache | Meaning |
|------------|---|
| 0b000 | Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability. |
| 0b001 | Normal Outer Non-cacheable. |
| 0b010 | Normal Outer Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Outer Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Outer Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Outer Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-back. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [55:52]

Reserved, RES0.

Physical_Address, bits [51:12]

Bits [51:12] of the physical address containing the virtual LPI Configuration table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the LPI Configuration table.

| Shareability | Meaning |
|--------------|----------------------------|
| 0b00 | Non-shareable. |
| 0b01 | Inner Shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Reserved. Treated as 0b00. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

InnerCache, bits [9:7]

Indicates the Inner Cacheability attributes of accesses to the LPI Configuration table.

| InnerCache | Meaning |
|------------|--|
| 0b000 | Device-nGnRnE. |
| 0b001 | Normal Inner Non-cacheable. |
| 0b010 | Normal Inner Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Inner Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Inner Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Inner Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-back. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

IDbits, bits [4:0]

The number of bits of virtual LPI INTID supported, minus one.

If the value of this field is less than 0b1101, indicating that the largest INTID is less than 8192 (the smallest LPI interrupt ID), the GIC will behave as if all virtual LPIs are out of range.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

When FEAT_GICv4p1 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------|------|------------|------------|----------|-----------|----|------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------|------------|------|----|----|----|----|----|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| Valid | RES0 | Entry_Size | OuterCache | Indirect | Page_Size | Z | Physical Address | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Physical Address | | | | | | | | | | | | | | | | | | | | | Shareability | InnerCache | Size | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Valid, bit [63]

This bit controls whether the vPE Configuration Table is valid.

| Valid | Meaning |
|-------|---|
| 0b0 | The vPE Configuration table is not valid. |
| 0b1 | The vPE Configuration table is valid. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Bit [62]

Reserved, RES0.

Entry_Size, bits [61:59]

Specifies the number 64-bit doublewords per table entry, minus one.

This bit is read-only.

OuterCache, bits [58:56]

Indicates the Outer Cacheability attributes of accesses to the table.

| OuterCache | Meaning |
|------------|---|
| 0b000 | Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability. |
| 0b001 | Normal Outer Non-cacheable. |
| 0b010 | Normal Outer Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Outer Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Outer Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Outer Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-back. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Indirect, bit [55]

This field indicates whether GICR_VPROPBASER specifies a single, flat table or a two-level table where the first level contains a list of descriptors.

| Indirect | Meaning |
|----------|--|
| 0b0 | Single Level. The Size field indicates the number of pages used to store data associated with each table entry. |
| 0b1 | Two Level. The Size field indicates the number of pages that contain an array of 64-bit descriptors to pages that are used to store the data associated with each table entry. A little endian memory order model is used. |

This field is RAZ/WI for GIC implementations that only support flat tables.

If the supported vPEID width indicated by [GICD_TYPER2.VIL](#) and [GICD_TYPER2.VID](#), and the smallest page size that is supported result in a single level table that requires multiple pages, then implementing this bit as RAZ/WI is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Page_Size, bits [54:53]

The following values indicate the size of page that the translation table uses:

| Page_Size | Meaning |
|-----------|----------------------------|
| 0b00 | 4KB. |
| 0b01 | 16KB. |
| 0b10 | 64KB. |
| 0b11 | Reserved. Treated as 0b10. |

Note

If the GIC implementation supports only a single, fixed page size, this field might be RO.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Z, bit [52]

When GICR_VPROPBASER.Valid is written from 0 to 1, GICR_VPROPBASER.Z indicates whether the vPE Configuration table is known to contain all zeros.

| Z | Meaning |
|-----|--|
| 0b0 | The vPE Configuration table is not zero, and contains live data. |
| 0b1 | The vPE Configuration table is zero. |

Setting GICR_VPROPBASER.Z to 0 causes the IRI to reload configuration from memory

When GICR_VPROPBASER.Valid is written from 0 to 1, if GICR_VPROPBASER.Z==1 behavior is UNPREDICTABLE if the allocated memory does not contain all zeros.

This field is WO, and reads as 0.

Physical_Address, bits [51:12]

Bits [51:12] of the physical address containing the LPI Configuration table.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the LPI Configuration table.

| Shareability | Meaning |
|--------------|----------------------------|
| 0b00 | Non-shareable. |
| 0b01 | Inner Shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Reserved. Treated as 0b00. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

InnerCache, bits [9:7]

Indicates the Inner Cacheability attributes of accesses to the LPI Configuration table.

| InnerCache | Meaning |
|------------|--|
| 0b000 | Device-nGnRnE. |
| 0b001 | Normal Inner Non-cacheable. |
| 0b010 | Normal Inner Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Inner Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Inner Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Inner Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-back. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Size, bits [6:0]

The number of pages of physical memory allocated to the table, minus one.

[GICR_VPROPBASER](#).Page_Size specifies the size of each page.

The reset behavior of this field is:

- On a GIC reset, this field resets to an UNKNOWN value.

Accessing GICR_VPROPBASER

GICR_VPROPBASER can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|-----------|--------|-----------------|
| GIC Redistributor | VLPI_base | 0x0070 | GICR_VPROPBASER |

Accesses on this interface are **RW**.

GICR_VSGIPENDR, Redistributor virtual SGI pending state register

The GICR_VSGIPENDR characteristics are:

Purpose

Requests the pending state of virtual SGIs for a specified vPE.

Configuration

This register is present only when FEAT_GICv4p1 is implemented. Otherwise, direct accesses to GICR_VSGIPENDR are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_VSGIPENDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Busy | | RES0 | | | | | | | | | | | | | | Pending | | | | | | | | | | | | | | | |

Busy, bit [31]

ID of target vPEID

| Busy | Meaning |
|------|---|
| 0b0 | Query of virtual SGI state not in progress. |
| 0b1 | Query of virtual SGI state in progress. |

Bits [30:16]

Reserved, RES0.

Pending, bits [15:0]

Pending state of virtual SGIs for requested vPEID.

This field is UNKNOWN when [GICR_VSGIPENDR](#).Busy == 1

Accessing GICR_VSGIPENDR

64-bit access only.

GICR_VSGIPENDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|-----------|--------|----------------|
| GIC Redistributor | VLPI_base | 0x0088 | GICR_VSGIPENDR |

Accesses on this interface are **RO**.

GICR_VSGIR, Redistributor virtual SGI pending state request register

The GICR_VSGIR characteristics are:

Purpose

Requests the pending state of virtual SGIs for a specified vPE.

Configuration

This register is present only when FEAT_GICv4p1 is implemented. Otherwise, direct accesses to GICR_VSGIR are RES0.

A copy of this register is provided for each Redistributor.

Attributes

GICR_VSGIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | vPEID | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

vPEID, bits [15:0]

ID of target vPE

Writing this field is CONSTRAINED UNPREDICTABLE when [GICR_VSGIPENDR](#).Busy == 1, with either the write ignored or a new query started.

Writing a value greater than the configured vPEID width behaviour is CONSTRAINED UNPREDICTABLE:

- GICR_VPENDBASER.vPEID is treated as having an UNKNOWN valid value for all purposes other than a direct read of the register.
- GICR_VPENDBASER.Valid is treated as being set to 0 for all purposes other than a direct read of the register.

The size of this field is IMPLEMENTATION DEFINED, and is specified by the [GICD_TYPER2](#).VIL and [GICD_TYPER2](#).VID fields. Unimplemented bits are RES0.

Accessing GICR_VSGIR

64-bit access only.

GICR_VSGIR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|-------|--------|----------|
|-----------|-------|--------|----------|

GICR_VSGIR, Redistributor virtual SGI pending state request register

| | | | |
|----------------------|-----------|--------|------------|
| GIC Redistributor | VLPI_base | 0x0080 | GICR_VSGIR |
|----------------------|-----------|--------|------------|

Accesses on this interface are **WO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICR_WAKER, Redistributor Wake Register

The GICR_WAKER characteristics are:

Purpose

Permits software to control the behavior of the WakeRequest power management signal corresponding to the Redistributor. Power management operations follow the rules in 'Power management' in in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Configuration

A copy of this register is provided for each Redistributor.

Attributes

GICR_WAKER is a 32-bit register.

Field descriptions

| | | | | |
|---------------------------|---|----------------|----------------|---------------------------|
| 31 | 3029282726252423222120191817161514131211109876543 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | RES0 | ChildrenAsleep | ProcessorSleep | IMPLEMENTATION DEFINED |

IMPLEMENTATION DEFINED, bit [31]

IMPLEMENTATION DEFINED.

Bits [30:3]

Reserved, RES0.

ChildrenAsleep, bit [2]

Read-only. Indicates whether the connected PE is quiescent:

| ChildrenAsleep | Meaning |
|----------------|---|
| 0b0 | An interface to the connected PE might be active. |
| 0b1 | All interfaces to the connected PE are quiescent. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 1.

ProcessorSleep, bit [1]

Indicates whether the Redistributor can assert the **WakeRequest** signal:

| ProcessorSleep | Meaning |
|----------------|---|
| 0b0 | This PE is not in, and is not entering, a low power state. |
| 0b1 | <p>The PE is either in, or is in the process of entering, a low power state.</p> <p>All interrupts that arrive at the Redistributor:</p> <ul style="list-style-type: none"> • Assert a WakeRequest signal. • Are held in the pending state at the Redistributor, and are not communicated to the CPU interface. <hr/> <p>Note</p> <p>When ProcessorSleep == 1, the Redistributor must ensure that any interrupts that are pending on the CPU interface are released.</p> <hr/> <p>For an implementation that is using the GIC Stream Protocol Interface:</p> <ul style="list-style-type: none"> • A Quiesce command puts the interface between the Redistributor and the CPU interface in a quiescent state. For more information, see 'Quiesce (IRI)' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069). • A Release command releases any interrupts that are pending on the CPU interface. For more information, see 'Release (ICC)' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069). |

Note

Before powering down a PE, software must set this bit to 1 and wait until ChildrenAsleep == 1. After powering up a PE, or following a failed powerdown, software must set this bit to 0 and wait until ChildrenAsleep == 0.

Changing ProcessorSleep from 1 to 0 when ChildrenAsleep is not 1 results in UNPREDICTABLE behavior.

Changing ProcessorSleep from 0 to 1 when the Enable for each interrupt group in the associated CPU interface is not 0 results in UNPREDICTABLE behavior.

The reset behavior of this field is:

- On a GIC reset, this field resets to 1.

IMPLEMENTATION DEFINED, bit [0]

IMPLEMENTATION DEFINED.

Accessing GICR_WAKER

To ensure a Redistributor is quiescent, software must write to GICR_WAKER with ProcessorSleep == 1, then poll the register until ChildrenAsleep == 1.

Resetting the connected PE when GICR_WAKER.ProcessorSleep==0 or GICR_WAKER.ChildrenAsleep==0, can lead to UNPREDICTABLE behaviour in the IRI.

Resetting the IRI when GICR_WAKER.ProcessorSleep==0 or GICR_WAKER.ChildrenAsleep==0 can lead to UNPREDICTABLE behaviour in the connected PE.

GICR_WAKER can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-------------------|---------|--------|------------|
| GIC Redistributor | RD_base | 0x0014 | GICR_WAKER |

This interface is accessible as follows:

- When GICD_CTLR.DS == 1 accesses to this register are **RW**.
- When GICD_CTLR.DS == 0 and an access is Secure accesses to this register are **RW**.
- When GICD_CTLR.DS == 0 and an access is Non-secure accesses to this register are **RAZ/WI**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRoot() accesses to this register are **RW**.
- When GICD_CTLR.DS == 0, FEAT_RME is implemented and IsAccessRealm() accesses to this register are **RAZ/WI**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_ABPR, Virtual Machine Aliased Binary Point Register

The GICV_ABPR characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 1 interrupt preemption.

This register corresponds to [GICC_ABPR](#) in the physical CPU interface.

Note

[GICH_LR<n>](#).Group determines whether a virtual interrupt is Group 0 or Group 1.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_ABPR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_ABPR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | Binary_Point | | | | | | | | | | | | | |

Bits [31:3]

Reserved, RES0.

Binary_Point, bits [2:0]

Controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

For information about how this field determines the interrupt priority bits assigned to the group priority field, see 'Priority grouping' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

The Binary_Point field of this register is aliased to [GICH_VMCR.VBPR1](#).

Accessing GICV_ABPR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_BPR1](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_BPR1_EL1](#) provides equivalent functionality.

The value contained in this register is one greater than the actual applied binary point value, as described in 'Priority grouping' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This register is used for Group 1 interrupts when [GICV_CTLR](#).CBPR == 0. [GICV_BPR](#) provides equivalent functionality for Group 0 interrupts, and for Group 1 interrupts when [GICV_CTLR](#).CBPR == 1.

GICV_ABPR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|-----------|
| GIC Virtual CPU interface | 0x001C | GICV_ABPR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_AEOIR, Virtual Machine Aliased End Of Interrupt Register

The GICV_AEOIR characteristics are:

Purpose

A write to this register performs a priority drop for the specified Group 1 virtual interrupt and, if [GICV_CTLR.EOImode](#) == 0, also deactivates the interrupt.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_AEOIR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_AEOIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

A successful EOI request means that:

- The highest priority bit in [GICH_APR<n>](#) is cleared, causing the running priority to drop.
- If the appropriate [GICV_CTLR.EOImode](#) bit == 0, the interrupt is deactivated in the corresponding List register. If the INTID corresponds to a hardware interrupt, the interrupt is also deactivated in the Distributor.

Note

Only Group 1 interrupts can target the hypervisor, and therefore only Group 1 interrupts are deactivated in the Distributor.

A write to this register is UNPREDICTABLE if the INTID corresponds to a Group 0 interrupt. In addition, the following GICv2 UNPREDICTABLE cases require specific actions:

- If highest active priority is Group 0 and the identified interrupt is in the List Registers and it matches the highest active priority. When EL2 is using System registers and [ICH_VTR_EL2](#).SEIS is 1, an IMPLEMENTATION DEFINED SEI might be generated, otherwise GICv3 implementations must ignore such writes.
- If the identified interrupt is in the List Registers, and the HW bit is 1, and the interrupt to be deactivated is an SGI (that is, the value of Physical_ID is between 0 and 15). GICv3 implementations must perform the deactivate operation. This means that a GICv3 implementation in legacy operation must ensure only a single SGI is active for a PE.
- If the identified interrupt is in the List Registers, and the HW bit is 1, and the corresponding pINTID field value is between 1020 and 1023, indicating a special purpose INTID. GICv3 implementations must not perform a deactivate operation but must still change the state of the List register as appropriate. When EL2 is using System registers and [ICH_VTR_EL2](#).SEIS is 1, an implementation might generate a system error.

Accessing GICV_AEOIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_EOIR1](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_EOIR1_EL1](#) provides equivalent functionality.

This register is used for Group 1 interrupts only. [GICV_EOIR](#) provides equivalent functionality for Group 0 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_AEOIR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|------------|
| GIC Virtual CPU interface | 0x0024 | GICV_AEOIR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **WO**.
- When an access is Secure accesses to this register are **WO**.
- When an access is Non-secure accesses to this register are **WO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_AHPPIR, Virtual Machine Aliased Highest Priority Pending Interrupt Register

The GICV_AHPPIR characteristics are:

Purpose

Provides the INTID of the highest priority pending Group 1 virtual interrupt in the List registers.

This register corresponds to the physical CPU interface register [GICC_AHPPIR](#).

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_AHPPIR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_AHPPIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

A read of this register returns the spurious INTID 1023 if any of the following are true:

- There are no pending interrupts of sufficiently high priority value to be signaled to the PE.
- The highest priority pending interrupt is in Group 0.

Accessing GICV_AHPPIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_HPPIR1](#) provides equivalent functionality.

- For AArch64 implementations, [ICC_HPPIR1_EL1](#) provides equivalent functionality.

This register is used for Group 1 interrupts only. [GICV_HPPIR](#) provides equivalent functionality for Group 0 interrupts.

The register does not return the INTID of an interrupt that is active and pending.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_AHPPIR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|-------------|
| GIC Virtual CPU interface | 0x0028 | GICV_AHPPIR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_AIAR, Virtual Machine Aliased Interrupt Acknowledge Register

The GICV_AIAR characteristics are:

Purpose

Provides the INTID of the signaled Group 1 virtual interrupt. A read of this register by the PE acts as an acknowledge for the interrupt.

This register corresponds to the physical CPU interface register [GICC_AIAR](#).

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_AIAR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_AIAR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

The operation of this register is similar to the operation of [GICV_IAR](#). When a vPE reads this register, the corresponding [GICH_LR<n>](#).Group field is checked to determine whether the interrupt is in Group 0 or Group 1:

- If the interrupt is Group 0, the spurious INTID 1023 is returned and the interrupt is not acknowledged.
- If the interrupt is Group 1, the INTID is returned. The List register entry is updated to active state, and the appropriate bit in [GICH_APR<n>](#) is set to 1.

A read of this register returns the spurious INTID 1023 if any of the following are true:

- When the virtual CPU interface is enabled and [GICH_HCR](#).En == 1:

- There are no pending interrupts of sufficiently high priority value to be signaled to the PE.
- The highest priority pending interrupt is in Group 0.
- Interrupt signaling by the virtual CPU interface is disabled.

Accessing GICV_AIAR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_IAR1](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_IAR1_EL1](#) provides equivalent functionality.

This register is used for Group 1 interrupts only. [GICV_IAR](#) provides equivalent functionality for Group 0 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_AIAR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|-----------|
| GIC Virtual CPU interface | 0x0020 | GICV_AIAR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_APR<n>, Virtual Machine Active Priorities Registers, n = 0 - 3

The GICV_APR<n> characteristics are:

Purpose

Provides information about interrupt active priorities.
These registers correspond to the physical CPU interface registers [GICC_APR<n>](#).

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_APR<n> are RES0.
When System register access is disabled for EL2, these registers access [GICH_APR<n>](#), and all active priorities for virtual machines are held in [GICH_APR<n>](#) regardless of interrupt group.
When System register access is enabled for EL2, these registers access [ICH_AP1R<n>_EL2](#), and all active priorities for virtual machines are held in [ICH_AP1R<n>_EL2](#) regardless of interrupt group.

Attributes

GICV_APR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|---------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|--------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P31 | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

P<x>, bit [x], for x = 31 to 0

Provides information about active priorities for the virtual machine.
See [GICH_APR<n>](#) and [ICH_AP1R<n>_EL2](#) for the correspondence between priorities and bits.

Accessing GICV_APR<n>

If System register access is not enabled for EL2, these registers access [GICH_APR<n>](#). If System register access is enabled for EL2, these registers access [ICH_AP1R<n>_EL2](#). All active priority mapped guests are held in the accessed registers, regardless of interrupt group.

GICV_APR<n> can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|------------------|-------------|
| GIC Virtual CPU interface | 0x00D0 + (4 * n) | GICV_APR<n> |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

GICV_BPR, Virtual Machine Binary Point Register

The GICV_BPR characteristics are:

Purpose

Defines the point at which the priority value fields split into two parts, the group priority field and the subpriority field. The group priority field determines Group 0 interrupt preemption.

This register corresponds to [GICC_BPR](#) in the physical CPU interface.

Note

[GICH_LR<n>](#).Group determines whether a virtual interrupt is Group 0 or Group 1.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_BPR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

When [GICV_CTLR](#).CBPR == 1, this register determines interrupt preemption for both Group 0 and Group 1 interrupts.

Attributes

GICV_BPR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | Binary Point | | | | | | | | | | | | | |

Bits [31:3]

Reserved, RES0.

Binary_Point, bits [2:0]

Controls how the 8-bit interrupt priority field is split into a group priority field, that determines interrupt preemption, and a subpriority field.

For information about how this field determines the interrupt priority bits assigned to the group priority field, see 'ICC_BPR0_EL1 Binary Point for Group 1 interrupts when CBPR == 1, or for Group 0 interrupts' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

The Binary_Point field of this register is aliased to [GICH_VMCR](#).VBPR0.

Accessing GICV_BPR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_BPR0](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_BPR0_EL1](#) provides equivalent functionality.

GICV_BPR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|----------|
| GIC Virtual CPU interface | 0x0008 | GICV_BPR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_CTLR, Virtual Machine Control Register

The GICV_CTLR characteristics are:

Purpose

Controls the behavior of virtual interrupts.

This register corresponds to the physical CPU interface register [GICC_CTLR](#).

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_CTLR are RES0.

This register is available when a GIC implementation supports interrupt virtualization.

Attributes

GICV_CTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|---------|----|------|----|------|----|-------|----|--------|----|------------|----|------------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | EOImode | | RES0 | | CBPR | | FIQEn | | AckCtl | | EnableGrp1 | | EnableGrp0 | | | | | | | | | |

Bits [31:10]

Reserved, RES0.

EOImode, bit [9]

Controls the behavior associated with the [GICV_EOIR](#), [GICV_AEOIR](#), and [GICV_DIR](#) registers:

| EOImode | Meaning |
|---------|---|
| 0b0 | Writes to GICV_EOIR and GICV_AEOIR perform priority drop and deactivate interrupt operations simultaneously. Behavior on a write to GICV_DIR is unpredictable. When it has completed processing the interrupt, the virtual machine writes to GICV_EOIR or GICV_AEOIR to deactivate the interrupt. The write updates the List registers and causes the virtual CPU interface to signal the interrupt completion to the physical Distributor. |
| 0b1 | Writes to GICV_EOIR and GICV_AEOIR perform priority drop operation only. Writes to GICV_DIR perform deactivate interrupt operation only. When it has completed processing the interrupt, the virtual machine writes to GICV_DIR to deactivate the interrupt. The write updates the List registers and causes the virtual CPU interface to signal the interrupt completion to the Distributor. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Bits [8:5]

Reserved, RES0.

CBPR, bit [4]

Controls whether [GICV_BPR](#) affects both Group 0 and Group 1 interrupts:

| CBPR | Meaning |
|------|--|
| 0b0 | GICV_BPR affects Group 0 virtual interrupts only. GICV_ABPR affects Group 1 virtual interrupts only. |
| 0b1 | GICV_BPR affects both Group 0 and Group 1 virtual interrupts. |

For more information, see 'Priority grouping' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

FIQEn, bit [3]

FIQ Enable. Controls whether Group 0 virtual interrupts are presented as virtual FIQs:

| FIQEn | Meaning |
|-------|---|
| 0b0 | Group 0 virtual interrupts are presented as virtual IRQs. |
| 0b1 | Group 0 virtual interrupts are presented as virtual FIQs. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

AckCtl, bit [2]

Arm deprecates use of this bit. Arm strongly recommends that software is written to operate with this bit always cleared to 0.

Acknowledge control. When the highest priority interrupt is Group 1, determines whether [GICV_IAR](#) causes the CPU interface to acknowledge the interrupt or returns the spurious identifier 1022, and whether [GICV_HPPIR](#) returns the interrupt ID or the special identifier 1022.

| AckCtl | Meaning |
|--------|---|
| 0b0 | If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns an interrupt ID of 1022. |
| 0b1 | If the highest priority pending interrupt is Group 1, a read of GICV_IAR or GICV_HPPIR returns the interrupt ID of the corresponding interrupt. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EnableGrp1, bit [1]

Enables the signaling of Group 1 virtual interrupts by the virtual CPU interface to the virtual machine:

| EnableGrp1 | Meaning |
|------------|--|
| 0b0 | Signaling of Group 1 interrupts is disabled. |
| 0b1 | Signaling of Group 1 interrupts is enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

EnableGrp0, bit [0]

Enables the signaling of Group 0 virtual interrupts by the virtual CPU interface to the virtual machine:

| EnableGrp0 | Meaning |
|------------|--|
| 0b0 | Signaling of Group 0 interrupts is disabled. |
| 0b1 | Signaling of Group 0 interrupts is enabled. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GICV_CTLR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_CTLR](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_CTLR_EL1](#) provides equivalent functionality.

GICV_CTLR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|-----------|
| GIC Virtual CPU interface | 0x0000 | GICV_CTLR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_DIR, Virtual Machine Deactivate Interrupt Register

The GICV_DIR characteristics are:

Purpose

Deactivates a specified virtual interrupt in the [GICH_LR<n>](#) List registers.

This register corresponds to the physical CPU interface register [GICC_DIR](#).

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_DIR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_DIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

When the virtual machine writes to this register, the specified interrupt in the List registers is changed from active to inactive, or from active and pending to pending. If the specified interrupt is present in the List registers but is not in either the active or active and pending states, the effect is UNPREDICTABLE. If the specified interrupt is not present in the List registers, [GICH_HCR](#).EOICount is incremented, potentially generating a maintenance interrupt.

Note

If the specified interrupt is not present in the List registers, the virtual machine cannot recover the INTID. Therefore, the hypervisor must ensure that, when [GICV_CTLR](#).EOImode == 1, no more than one active interrupt is

transferred from the List registers into a software list. If more than one active interrupt that is not stored in the List registers exists, the hypervisor must handle accesses to GICV_DIR in software, typically by trapping these accesses.

If the corresponding [GICH_LR<n>.HW == 1](#), indicating a hardware interrupt, then a deactivate request is sent to the physical Distributor, identifying the physical INTID from the corresponding field in the List register. This effect is identical to a Non-secure write to [GICC_DIR](#) from the PE having that physical INTID. This means that if the corresponding physical interrupt is marked as Group 0, the request is ignored.

Note

Interrupt deactivation using this register is based on the provided INTID, with no requirement to deactivate interrupts in any particular order. A single register is therefore used to deactivate both Group 0 and Group 1 interrupts.

Accessing GICV_DIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_DIR](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_DIR_EL1](#) provides equivalent functionality.

Writes to this register are valid only when [GICV_CTLR.EOImode == 1](#). Writes to this register are otherwise UNPREDICTABLE.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_DIR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|----------|
| GIC Virtual CPU interface | 0x1000 | GICV_DIR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **WO**.
- When an access is Secure accesses to this register are **WO**.
- When an access is Non-secure accesses to this register are **WO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_EOIR, Virtual Machine End Of Interrupt Register

The GICV_EOIR characteristics are:

Purpose

A write to this register performs a priority drop for the specified Group 0 virtual interrupt and, if [GICV_CTLR.EOImode](#) == 0, also deactivates the interrupt.

This register corresponds to the physical CPU interface register [GICC_EOIR](#).

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_EOIR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_EOIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

The behavior of this register depends on the setting of [GICV_CTLR.EOImode](#):

| GICV_CTLR.EOImode | Behavior |
|-----------------------------------|---|
| 0b0 | Both the priority drop and the deactivate interrupt effects occur |
| 0b1 | Only the priority drop effect occurs. |

A successful EOI request means that:

- The highest priority bit in [GICH_APR<n>](#) is cleared, causing the running priority to drop.
- If the appropriate [GICV_CTLR.EOImode](#) bit == 0, the interrupt is deactivated in the corresponding List register [GICH_LR<n>](#). If [GICH_LR<n>.HW](#) == 1, indicating the INTID corresponds to a hardware interrupt,

a deactivate request is also sent to the physical Distributor, identifying the physical INTID from the corresponding field in the List register. This effect is identical to a Non-secure write to [GICC_DIR](#) from the PE having that physical INTID. This means that if the corresponding physical interrupt is marked as Group 0, and [GICD_CTLR.DS](#) == 0, the deactivation request is ignored. See [GICC_EOIR](#) for more information.

Note

Only Group 1 interrupts can target the hypervisor, and therefore only Group 1 interrupts are deactivated in the Distributor.

Accessing GICV_EOIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_EOIR0](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_EOIR0_EL1](#) provides equivalent functionality.

This register is used for Group 0 interrupts only. [GICV_AEOIR](#) provides equivalent functionality for Group 1 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_EOIR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|-----------|
| GIC Virtual CPU interface | 0x0010 | GICV_EOIR |

This interface is accessible as follows:

- When [GICD_CTLR.DS](#) == 0 accesses to this register are **WO**.
- When an access is Secure accesses to this register are **WO**.
- When an access is Non-secure accesses to this register are **WO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_HPPIR, Virtual Machine Highest Priority Pending Interrupt Register

The GICV_HPPIR characteristics are:

Purpose

Provides the INTID of the highest priority pending Group 0 virtual interrupt in the List registers.

This register corresponds to the physical CPU interface register [GICC_HPPIR](#).

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_HPPIR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_HPPIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

Reads of the GICC_HPPIR that do not return a valid INTID return a spurious INTID, 1022 or 1023. See 'Special INTIDs' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

| Highest priority pending interrupt Group | GICV_HPPIR read | GICV_CTLR.AckCtl | Returned INTID |
|--|-----------------|----------------------------------|-------------------------|
| 1 | Non-secure | x | ID of Group 1 interrupt |
| 1 | Secure | 0 | 1022 |
| 1 | Secure | 1 | ID of Group 1 interrupt |
| 0 | Non-secure | x | 1023 |
| 0 | Secure | x | ID of Group 0 interrupt |
| No pending interrupts | x | x | 1023 |

If the CPU interface supports only a single Security state, the entries that apply to Secure reads describe the behavior.

Accessing GICV_HPPIR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_HPPIR0](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_HPPIR0_EL1](#) provides equivalent functionality.

This register is used for Group 0 interrupts only. [GICV_AHPPIR](#) provides equivalent functionality for Group 1 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_HPPIR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|------------|
| GIC Virtual CPU interface | 0x0018 | GICV_HPPIR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

GICV_IAR, Virtual Machine Interrupt Acknowledge Register

The GICV_IAR characteristics are:

Purpose

Provides the INTID of the signaled Group 0 virtual interrupt. A read of this register by the PE acts as an acknowledge for the interrupt.

This register corresponds to the physical CPU interface register [GICC_IAR](#).

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_IAR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_IAR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | INTID | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:25]

Reserved, RES0.

INTID, bits [24:0]

The INTID of the signaled interrupt.

Note

INTIDs 1020-1023 are reserved and convey additional information such as spurious interrupts.

When affinity routing is not enabled:

- Bits [23:13] are RES0.
- For SGIs, bits [12:10] identify the CPU interface corresponding to the source PE. For all other interrupts these bits are RES0.

When the virtual machine writes to this register, the virtual CPU interface acknowledges the highest priority pending virtual interrupt and sets the state in the corresponding List register to active. The appropriate bit in the active priorities register [GICH_APR<n>](#) is set to 1.

If [GICH_LR<n>](#).HW == 0, indicating that the interrupt is software-triggered, then bits [12:10] of [GICH_LR<n>](#) are returned in bits [12:10] of GICV_IAR. Otherwise bits [12:10] are RES0.

A read of this register returns the spurious INTID 1023 if either of the following is true:

- There are no pending interrupts of sufficiently high priority value to be signaled to the PE with the virtual CPU interface enabled and [GICH_HCR](#).En == 1.
- Interrupt signaling by the virtual CPU interface is disabled.

A read of this register returns the spurious INTID 1022 if the highest priority pending interrupt is Group 1 and [GICV_CTLR](#).AckCtl == 0.

Accessing GICV_IAR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_IAR0](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_IAR0_EL1](#) provides equivalent functionality.

This register is used for Group 0 interrupts only. [GICV_AIAR](#) provides equivalent functionality for Group 1 interrupts.

When affinity routing is enabled, it is a programming error to use memory-mapped registers to access the GIC.

GICV_IAR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|----------|
| GIC Virtual CPU interface | 0x000C | GICV_IAR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_IIDR, Virtual Machine CPU Interface Identification Register

The GICV_IIDR characteristics are:

Purpose

Provides information about the implementer and revision of the virtual CPU interface.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_IIDR are RES0.

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states this register is Common.

Attributes

GICV_IIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|----------------------|----|----|----|----------|----|----|----|-------------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ProductID | | | | | | | | | | | | Architecture_version | | | | Revision | | | | Implementer | | | | | | | | | | | |

ProductID, bits [31:20]

Product Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Architecture_version, bits [19:16]

The version of the GIC architecture that is implemented.

| Architecture_version | Meaning |
|----------------------|---|
| 0b0001 | GICv1. |
| 0b0010 | GICv2. |
| 0b0011 | GICv3 memory-mapped interface supported. Support for the System register interface is discoverable from PE registers ID_PFR1 and ID_AA64PFR0_EL1. |
| 0b0100 | GICv4 memory-mapped interface supported. Support for the System register interface is discoverable from PE registers ID_PFR1 and ID_AA64PFR0_EL1. |

Other values are reserved.

Revision, bits [15:12]

Revision number for the CPU interface.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 code of the company that implemented the CPU interface.

- Bits [11:8] are the JEP106 continuation code of the implementer. For an Arm implementation, this field is 0x4.
- Bit [7] is always 0.
- Bits [6:0] are the JEP106 identity code of the implementer. For an Arm implementation, bits [7:0] are therefore 0x3B.

Accessing GICV_IIDR

GICV_IIDR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|-----------|
| GIC Virtual CPU interface | 0x00FC | GICV_IIDR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_PMR, Virtual Machine Priority Mask Register

The GICV_PMR characteristics are:

Purpose

This register provides a virtual interrupt priority filter. Only virtual interrupts with a higher priority than the value in this register are signaled to the PE.

Note

Higher interrupt priority corresponds to a lower value of the Priority field.

This register corresponds to the physical CPU interface register [GICC_PMR](#).

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_PMR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

The Priority field of this register is aliased to [GICH_VMCR.VMPR](#), to enable state to be switched easily between virtual machines during context-switching.

Attributes

GICV_PMR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|----------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | Priority | | | | | | | |

Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The priority mask level for the virtual CPU interface. If the priority of the interrupt is higher than the value indicated by this field, the interface signals the interrupt to the PE.

If the GIC implementation supports fewer than 256 priority levels some bits might be RAZ/WI, as follows:

- For 128 supported levels, bit [0] = 0b0.
- For 64 supported levels, bits [1:0] = 0b00.
- For 32 supported levels, bits [2:0] = 0b000.
- For 16 supported levels, bits [3:0] = 0b0000.

For more information, see 'Interrupt prioritization' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing GICV_PMR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_PMR](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_PMR_EL1](#) provides equivalent functionality.

GICV_PMR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|----------|
| GIC Virtual CPU interface | 0x0004 | GICV_PMR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_RPR, Virtual Machine Running Priority Register

The GICV_RPR characteristics are:

Purpose

This register indicates the running priority of the virtual CPU interface.

This register corresponds to the physical CPU interface register [GICC_RPR](#).

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_RPR are RES0.

This register is available when the GIC implementation supports interrupt virtualization.

Attributes

GICV_RPR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|----------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | Priority | | | | | | | |

Bits [31:8]

Reserved, RES0.

Priority, bits [7:0]

The current running priority on the virtual CPU interface. This is the group priority of the current active interrupt.

If there are no active interrupts on the CPU interface, or all active interrupts have undergone a priority drop, the value returned is the Idle priority.

The priority returned is the group priority as if the BPR was set to the minimum value.

Accessing GICV_RPR

This register is used only when System register access is not enabled. When System register access is enabled:

- For AArch32 implementations, [ICC_RPR](#) provides equivalent functionality.
- For AArch64 implementations, [ICC_RPR_EL1](#) provides equivalent functionality.

Depending on the implementation, if no bits are set to 1 in [GICH_APR<n>](#), indicating no active virtual interrupts in the virtual CPU interface, the priority reads as 0xFF or 0xF8 to reflect the number of supported interrupt priority bits defined by [GICH_VTR](#).PRIbits.

GICV_RPR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|----------|
| GIC Virtual CPU interface | 0x0014 | GICV_RPR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RO**.
- When an access is Secure accesses to this register are **RO**.
- When an access is Non-secure accesses to this register are **RO**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GICV_STATUSR, Virtual Machine Error Reporting Status Register

The GICV_STATUSR characteristics are:

Purpose

Provides software with a mechanism to detect:

- Accesses to reserved locations.
- Writes to read-only locations.
- Reads of write-only locations.

Configuration

This register is present only when FEAT_GICv3_LEGACY is implemented and EL2 is implemented. Otherwise, direct accesses to GICV_STATUSR are RES0.

In systems where this register is implemented, Arm expects that when a virtual machine is scheduled, the hypervisor ensures that this register is cleared to 0. The hypervisor might check for illegal accesses when the virtual machine is unscheduled.

Attributes

GICV_STATUSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|------|------|-----|-----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | WROD | RWOD | WRD | RRD |

Bits [31:4]

Reserved, RES0.

WROD, bit [3]

Write to an RO location.

| WROD | Meaning |
|------|--|
| 0b0 | Normal operation. |
| 0b1 | A write to an RO location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

RWOD, bit [2]

Read of a WO location.

| RWOD | Meaning |
|------|--|
| 0b0 | Normal operation. |
| 0b1 | A read of a WO location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

WRD, bit [1]

Write to a reserved location.

| WRD | Meaning |
|-----|---|
| 0b0 | Normal operation. |
| 0b1 | A write to a reserved location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

RRD, bit [0]

Read of a reserved location.

| RRD | Meaning |
|-----|--|
| 0b0 | Normal operation. |
| 0b1 | A read of a reserved location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

Accessing GICV_STATUSR

This is an optional register. If the register is implemented, [GICC_STATUSR](#) must also be implemented. If the register is not implemented, the location is RAZ/WI.

This register is used only when System register access is not enabled. If System register access is enabled, this register is not updated. Equivalent function might be provided by appropriate traps and exceptions.

GICV_STATUSR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------------|--------|--------------|
| GIC Virtual CPU interface | 0x002C | GICV_STATUSR |

This interface is accessible as follows:

- When GICD_CTLR.DS == 0 accesses to this register are **RW**.
- When an access is Secure accesses to this register are **RW**.
- When an access is Non-secure accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_BASER<n>, ITS Translation Table Descriptors, n = 0 - 7

The GITS_BASER<n> characteristics are:

Purpose

Specifies the base address and size of the ITS translation tables.

Configuration

A copy of this register is provided for each ITS translation table.

Bits [63:32] and bits [31:0] are accessible independently.

A maximum of 8 GITS_BASER<n> registers can be provided. Unimplemented registers are RES0.

When [GITS_CTLR.Enabled](#) == 1 or [GITS_CTLR.Quiescent](#) == 0, writing this register is UNPREDICTABLE.

Attributes

GITS_BASER<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------|----------|------------|------|------------|------------|------------------|----|----|----|----|----|----|----|----|----|--------------|----|-----------|----|------|----|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| Valid | Indirect | InnerCache | Type | OuterCache | Entry_Size | Physical_Address | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Physical_Address | | | | | | | | | | | | | | | | Shareability | | Page_Size | | Size | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

Valid, bit [63]

Indicates whether software has allocated memory for the translation table:

| Valid | Meaning |
|-------|--|
| 0b0 | No memory is allocated for the translation table. The ITS discards any writes to the interrupt translation page when either: <ul style="list-style-type: none"> GITS_BASER<n>.Type specifies any valid table entry type other than interrupt collections, that is, any value other than 0b100. GITS_BASER<n>.Type specifies an interrupt collection and GITS_TYPER.HCC == 0. |
| 0b1 | Memory is allocated to the translation table. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Indirect, bit [62]

This field indicates whether an implemented register specifies a single, flat table or a two-level table where the first level contains a list of descriptors.

| Indirect | Meaning |
|----------|---|
| 0b0 | Single Level. The Size field indicates the number of pages used by the ITS to store data associated with each table entry. |
| 0b1 | Two Level. The Size field indicates the number of pages which contain an array of 64-bit descriptors to pages that are used to store the data associated with each table entry. A little endian memory order model is used. |

For more information, see 'The ITS tables' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

This field is RAZ/WI for GIC implementations that only support flat tables. If the maximum width of the scaling factor that is identified by GITS_BASER<n>.Type and the smallest page size that is supported result in a single level table that requires multiple pages, then implementing this bit as RAZ/WI is DEPRECATED.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

InnerCache, bits [61:59]

Indicates the Inner Cacheability attributes of accesses to the table. The possible values of this field are:

| InnerCache | Meaning |
|------------|--|
| 0b000 | Device-nGnRnE. |
| 0b001 | Normal Inner Non-cacheable. |
| 0b010 | Normal Inner Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Inner Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Inner Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Inner Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-back. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Type, bits [58:56]

Read only. Specifies the type of entity that requires entries in the corresponding translation table. The possible values of the field are:

| Type | Meaning |
|-------|--|
| 0b000 | Unimplemented. This register does not correspond to a translation table. |
| 0b001 | Devices. This register corresponds to a translation table that scales with the width of the DeviceID. Only a single GITS_BASER<n> register reports this type. |
| 0b010 | vPEs. FEAT_GICv4 only. This register corresponds to a translation table that scales with the number of vPEs in the system. The translation table requires (ENTRY_SIZE * N) bytes of memory, where N is the number of vPEs in the system. Only a single GITS_BASER<n> register reports this type. |
| 0b100 | Interrupt collections. This register corresponds to a translation table that scales with the number of interrupt collections in the system. The translation table requires (ENTRY_SIZE * N) bytes of memory, where N is the number of interrupt collections. Not more than one GITS_BASER<n> register will report this type. |

Other values are reserved.

For FEAT_GICv4p1, the registers are allocated as follows:

- GITS_BASER0.Type is 0b001 (Device).
- GITS_BASER1.Type is either 0b100 (Collection Table) or 0b000 (Unimplemented).
- GITS_BASER2.Type is either 0b010 (vPE) or 0b000 (Unimplemented).
- GITS_BASER<n>.Type, where 'n' is in the range 3 to 7, is 0b000 (Unimplemented).

For FEAT_GICv3, FEAT_GICv3p1, and FEAT_GICv4, Arm recommends that the GITS_BASER<n> use the same allocations.

Other allocations of Type values are deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

OuterCache, bits [55:53]

Indicates the Outer Cacheability attributes of accesses to the table. The possible values of this field are:

| OuterCache | Meaning |
|------------|---|
| 0b000 | Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability. |
| 0b001 | Normal Outer Non-cacheable. |
| 0b010 | Normal Outer Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Outer Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Outer Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Outer Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-back. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Entry_Size, bits [52:48]

Read-only. Specifies the number of bytes per translation table entry, minus one.

Physical_Address, bits [47:12]

Physical Address. When Page_Size is 4KB or 16KB:

- Bits [51:48] of the base physical address are zero.
- This field provides bits[47:12] of the base physical address of the table.
- Bits[11:0] of the base physical address are zero.
- The address must be aligned to the size specified in the Page Size field. Otherwise the effect is CONSTRAINED UNPREDICTABLE, and can be one of the following:
 - Bits[X:12], where X is derived from the page size, are treated as zero.
 - The value of bits[X:12] are used when calculating the address of a table access.

When Page_Size is 64KB:

- Bits[47:16] of the register provide bits[47:16] of the base physical address of the table.
- Bits[15:12] of the register provide bits[51:48] of the base physical address of the table.
- Bits[15:0] of the base physical address are 0.

In implementations that support fewer than 52 bits of physical address, any unimplemented upper bits might be RAZ/WI.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the table. The possible values of this field are:

| Shareability | Meaning |
|--------------|----------------------------|
| 0b00 | Non-shareable. |
| 0b01 | Inner Shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Reserved. Treated as 0b00. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Page_Size, bits [9:8]

The size of page that the translation table uses:

| Page_Size | Meaning |
|-----------|----------------------------|
| 0b00 | 4KB. |
| 0b01 | 16KB. |
| 0b10 | 64KB. |
| 0b11 | Reserved. Treated as 0b10. |

Note

If the GIC implementation supports only a single, fixed page size, this field might be RO.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Size, bits [7:0]

The number of pages of physical memory allocated to the table, minus one. GITS_BASER<n>.Page_Size specifies the size of each page.

If GITS_BASER<n>.Type == 0, this field is RAZ/WI.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Accessing GITS_BASER<n>

GITS_BASER<n> can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------------|------------------|---------------|
| GIC ITS control | 0x0100 + (8 * n) | GITS_BASER<n> |

Accesses on this interface are **RW**.

GITS_CBASER, ITS Command Queue Descriptor

The GITS_CBASER characteristics are:

Purpose

Specifies the base address and size of the ITS command queue.

Configuration

Bits [63:32] and bits [31:0] are accessible separately.

Attributes

GITS_CBASER is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|------|------------|------|------------|------|------------------|----|----|----|----|--------------|----|----|----|----|----|----|----|----|----|----|------------------|------|----|----|----|----|----|----|----|----|--|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | | |
| Valid | RES0 | InnerCache | RES0 | OuterCache | RES0 | Physical_Address | | | | | | | | | | | | | | | | Physical_Address | | | | | | | | | | | | |
| | | | | | | | | | | | Shareability | | | | | | | | | | | RES0 | Size | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | |

Valid, bit [63]

Indicates whether software has allocated memory for the command queue:

| Valid | Meaning |
|-------|---|
| 0b0 | No memory is allocated for the command queue. |
| 0b1 | Memory is allocated to the command queue. |

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Bit [62]

Reserved, RES0.

InnerCache, bits [61:59]

Indicates the Inner Cacheability attributes of accesses to the command queue. The possible values of this field are:

| InnerCache | Meaning |
|------------|--|
| 0b000 | Device-nGnRnE. |
| 0b001 | Normal Inner Non-cacheable. |
| 0b010 | Normal Inner Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Inner Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Inner Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Inner Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Inner Cacheable Read-allocate, Write-allocate, Write-back. |

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [58:56]

Reserved, RES0.

OuterCache, bits [55:53]

Indicates the Outer Cacheability attributes of accesses to the command queue. The possible values of this field are:

| OuterCache | Meaning |
|------------|---|
| 0b000 | Memory type defined in InnerCache field. For Normal memory, Outer Cacheability is the same as Inner Cacheability. |
| 0b001 | Normal Outer Non-cacheable. |
| 0b010 | Normal Outer Cacheable Read-allocate, Write-through. |
| 0b011 | Normal Outer Cacheable Read-allocate, Write-back. |
| 0b100 | Normal Outer Cacheable Write-allocate, Write-through. |
| 0b101 | Normal Outer Cacheable Write-allocate, Write-back. |
| 0b110 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-through. |
| 0b111 | Normal Outer Cacheable Read-allocate, Write-allocate, Write-back. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bit [52]

Reserved, RES0.

Physical_Address, bits [51:12]

Bits [51:12] of the base physical address of the command queue. Bits [11:0] of the base address are 0.

In implementations supporting fewer than 52 bits of physical address, unimplemented upper bits are RES0.

If bits [15:12] are not all zeros, behavior is a CONSTRAINED UNPREDICTABLE choice:

- Bits [15:12] are treated as if all the bits are zero. The value read back from those bits is either the value written or zero.
- The result of the calculation of an address for a command queue read can be corrupted.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Shareability, bits [11:10]

Indicates the Shareability attributes of accesses to the command queue. The possible values of this field are:

| Shareability | Meaning |
|--------------|----------------------------|
| 0b00 | Non-shareable. |
| 0b01 | Inner Shareable. |
| 0b10 | Outer Shareable. |
| 0b11 | Reserved. Treated as 0b00. |

It is IMPLEMENTATION DEFINED whether this field has a fixed value or can be programmed by software. Implementing this field with a fixed value is deprecated.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [9:8]

Reserved, RES0.

Size, bits [7:0]

The number of 4KB pages of physical memory allocated to the command queue, minus one.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

The command queue is a circular buffer and wraps at Physical Address [47:0] + (4096 * (Size + 1)).

Note

When this register is successfully written, the value of [GITS_CREADR](#) is set to zero.

Accessing GITS_CBASER

When [GITS_CTLR.Enabled](#) == 1 or [GITS_CTLR.Quiescent](#) == 0, writing this register is UNPREDICTABLE.

GITS_CBASER can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------------|--------|-------------|
| GIC ITS control | 0x0080 | GITS_CBASER |

Accesses on this interface are **RW**.

GITS_CREADR, ITS Read Register

The GITS_CREADR characteristics are:

Purpose

Specifies the offset from [GITS_CBASER](#) where the ITS reads the next ITS command.

Configuration

This register is cleared to 0 when a value is written to [GITS_CBASER](#).

Bits [63:32] and bits [31:0] are accessible separately.

Attributes

GITS_CREADR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|---------|----|----|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | Offset | | | | | | | | | | | | RES0 | | | | Stalled | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:20]

Reserved, RES0.

Offset, bits [19:5]

Bits [19:5] of the offset from [GITS_CBASER](#). Bits [4:0] of the offset are zero.

Bits [4:1]

Reserved, RES0.

Stalled, bit [0]

Reports whether the processing of commands is stalled because of a command error.

| Stalled | Meaning |
|---------|--|
| 0b0 | ITS command queue is not stalled because of a command error. |
| 0b1 | ITS command queue is stalled because of a command error. |

For more information, see 'The ITS command interface' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

Accessing GITS_CREADR

GITS_CREADR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
|-----------|--------|----------|

GITS_CREADR, ITS Read Register

| | | |
|-----------------|--------|-------------|
| GIC ITS control | 0x0090 | GITS_CREADR |
|-----------------|--------|-------------|

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_CTLR, ITS Control Register

The GITS_CTLR characteristics are:

Purpose

Controls the operation of an ITS.

Configuration

The ITS_Number (bits [7:4]) and bit [1] fields apply only in FEAT_GICv4 implementations, and are RES0 in FEAT_GICv3 implementations.

Attributes

GITS_CTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|------------|----|----|----|------|------|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Quiescent | RES0 | | | | | | | | | | | | | | | | UMSIlrq | ITS_Number | | | | RES0 | ImDe | Enabled | | | | | | | |

Quiescent, bit [31]

Read-only. Indicates completion of all ITS operations when GITS_CTLR.Enabled == 0.

| Quiescent | Meaning |
|-----------|--|
| 0b0 | The ITS is not quiescent and cannot be powered down. |
| 0b1 | The ITS is quiescent and can be powered down. |

For the ITS to be considered inactive, there must be no transactions in progress. In addition, all operations required to ensure that mapping data is consistent with external memory must be complete.

Note

In distributed GIC implementations, this bit is set to 1 only after the ITS forwards any operations that have not yet been completed to the Redistributors and receives confirmation that all such operations have reached the appropriate Redistributor.

In FEAT_GICv3, FEAT_GICv3p1, and FEAT_GICv4, when GITS_CTLR.Enabled == 1, the value of GITS_CTLR.Quiescent is UNKNOWN.

In FEAT_GICv4p1, when GITS_CTLR.Enabled == 1, the value of GITS_CTLR.Quiescent reads as 1 until the write to Enabled has taken effect and then reads as 0.

The reset behavior of this field is:

- On a GIC reset, this field resets to 1.

Bits [30:9]

Reserved, RES0.

UMSIlrq, bit [8]

Unmapped MSI reporting interrupt enable.

| UMSlirq | Meaning |
|---------|--|
| 0b0 | The ITS does not assert an interrupt signal when GITS_STATUSR.UMSI is 1. |
| 0b1 | The ITS asserts an interrupt signal when GITS_STATUSR.UMSI is 1. |

If [GITS_TYPER.UMSlirq](#) is 0, this field is RES0.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

ITS_Number, bits [7:4]

In FEAT_GICv3 implementations this field is RES0.

In FEAT_GICv4 implementations with more than one ITS instance, this field indicates the ITS number for use with 'VMOVP GICv4.0' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

It is IMPLEMENTATION DEFINED whether this field is programmable or RO.

If this field is programmable, changing this field when [GITS_CTLR.Quiescent](#) == 0 or [GITS_CTLR.Enabled](#) == 1 is UNPREDICTABLE.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [3:2]

Reserved, RES0.

ImDe, bit [1]

In GICv3 implementations, this bit is RES0.

In GICv4 implementations, this bit is IMPLEMENTATION DEFINED.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Enabled, bit [0]

Controls whether the ITS is enabled:

| Enabled | Meaning |
|---------|--|
| 0b0 | The ITS is not enabled. Writes to GITS_TRANSLATER are ignored and no further command queue entries are processed. |
| 0b1 | The ITS is enabled. Writes to GITS_TRANSLATER result in interrupt translations and the command queue is processed. |

If a write to this register changes this field from 1 to 0, the ITS must ensure that both:

- Any caches containing mapping data are made consistent with external memory.
- [GITS_CTLR.Quiescent](#) == 0 until all caches are consistent with external memory.

Changing [GITS_CTLR.Enabled](#) from 0 to 1 when [GITS_CTLR.Quiescent](#) is 0 results in UNPREDICTABLE behavior.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Accessing GITS_CTLR

GITS_CTLR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------------|--------|-----------|
| GIC ITS control | 0x0000 | GITS_CTLR |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_CWRITER, ITS Write Register

The GITS_CWRITER characteristics are:

Purpose

Specifies the offset from [GITS_CBASER](#) where software writes the next ITS command.

Configuration

Bits [63:32] and bits [31:0] are accessible separately.

Attributes

GITS_CWRITER is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | Offset | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [63:20]

Reserved, RES0.

Offset, bits [19:5]

Bits [19:5] of the offset from [GITS_CBASER](#). Bits [4:0] of the offset are zero.

The reset behavior of this field is:

- On a GIC reset, this field resets to an architecturally UNKNOWN value.

Bits [4:1]

Reserved, RES0.

Retry, bit [0]

Writing this bit has the following effects:

| Retry | Meaning |
|---|--|
| 0b0 | No effect on the processing commands by the ITS. |
| 0b1 | Restarts the processing of commands by the ITS if it stalled because of a command error. |
| Note | |
| If the processing of commands is not stalled because of a command error, writing 1 to this bit has no effect. | |

When read, this bit is RES0.

For more information, see 'The ITS command interface' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

If GITS_CWRITER is written with a value outside of the valid range specified by [GITS_CBASER.Physical_Address](#) and [GITS_CBASER.Size](#), behavior is a CONSTRAINED UNPREDICTABLE choice, as follows:

- The command queue is considered invalid, and no further commands are processed until GITS_CWRITER is written with a value that is in the valid range.
- The value is treated as a valid UNKNOWN value.

An implementation might choose to report a system error in an IMPLEMENTATION DEFINED manner.

Accessing GITS_CWRITER

GITS_CWRITER can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------------|--------|--------------|
| GIC ITS control | 0x0088 | GITS_CWRITER |

Accesses on this interface are **RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_IIDR, ITS Identification Register

The GITS_IIDR characteristics are:

Purpose

Provides information about the implementer and revision of the ITS.

Configuration

This register is available in all configurations of the GIC. If the GIC implementation supports two Security states, this register is Common.

Attributes

GITS_IIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|------|----|----|----|---------|----|----|----|----------|----|----|----|-------------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ProductID | | | | | | | | RES0 | | | | Variant | | | | Revision | | | | Implementer | | | | | | | | | | | |

ProductID, bits [31:24]

Product Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Bits [23:20]

Reserved, RES0.

Variant, bits [19:16]

Variant number. Typically, this field is used to distinguish product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

Revision number. Typically, this field is used to distinguish minor revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 code of the company that implemented the ITS:

- Bits [11:8] are the JEP106 continuation code of the implementer. For an Arm implementation, this field is 0x4.
- Bit [7] is always 0.

- Bits [6:0] are the JEP106 identity code of the implementer. For an Arm implementation, bits [7:0] are therefore 0x3B.

Accessing GITS_IIDR

GITS_IIDR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------------|--------|-----------|
| GIC ITS control | 0x0004 | GITS_IIDR |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_MPAMIDR, Report maximum PARTID and PMG Register

The GITS_MPAMIDR characteristics are:

Purpose

Reports the maximum support PARTID and PMG values.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GITS_MPAMIDR are RES0.

A copy of this register is provided for each ITS.

When [GITS_TYPER](#).MPAM==0, this register is RES0.

Attributes

GITS_MPAMIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|-----------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | PMGmax | | | | | | | | PARTIDmax | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

PMGmax, bits [23:16]

Maximum PMG value supported.

PARTIDmax, bits [15:0]

Maximum PARTID value supported.

Accessing GITS_MPAMIDR

GITS_MPAMIDR can be accessed through the memory-mapped interfaces:

| Component | Offset |
|-----------------|--------|
| GIC ITS control | 0x0010 |

Accesses on this interface are **RO**.

GITS_MPIDR, Report ITS's affinity.

The GITS_MPIDR characteristics are:

Purpose

Reports ITS's affinity when the vPE Table is shared with Redistributors.

Configuration

This register is present only when FEAT_GICv4p1 is implemented. Otherwise, direct accesses to GITS_MPIDR are RES0.

A copy of this register is provided for each ITS.

When [GITS_TYPER](#).SVPET==0, this register is RES0.

Attributes

GITS_MPIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Aff3 | | | | | | | | Aff2 | | | | | | | | Aff1 | | | | | | | | RES0 | | | | | | | |

Aff3, bits [31:24]

The Affinity level 3 value for the ITS.

Aff2, bits [23:16]

The Affinity level 2 value for the ITS.

Aff1, bits [15:8]

The Affinity level 1 value for the ITS.

Bits [7:0]

Reserved, RES0.

Accessing GITS_MPIDR

GITS_MPIDR can be accessed through the memory-mapped interfaces:

| Component | Offset |
|-----------------|--------|
| GIC ITS control | 0x0018 |

Accesses on this interface are **RO**.

GITS_PARTIDR, Set PARTID and PMG Register

The GITS_PARTIDR characteristics are:

Purpose

Sets the PARTID and PMG values used for memory accesses by the ITS.

Configuration

This register is present only when FEAT_GICv3p1 is implemented. Otherwise, direct accesses to GITS_PARTIDR are RES0.

A copy of this register is provided for each ITS.

When [GITS_TYPER](#).MPAM==0, this register is RES0.

Attributes

GITS_PARTIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | PMG | | | | | | | | PARTID | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

PMG value used when ITS accesses memory.

It is IMPLEMENTATION DEFINED whether bits not needed to represent PMG values in the range 0 to PMG_MAX are stateful or RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

PARTID, bits [15:0]

PARTID value used when ITS accesses memory.

It is IMPLEMENTATION DEFINED whether bits not needed to represent PARTID values in the range 0 to PARTID_MAX are stateful or RES0.

The reset behavior of this field is:

- On a GIC reset, this field resets to 0.

Accessing GITS_PARTIDR

GITS_PARTIDR can be accessed through the memory-mapped interfaces:

| Component | Offset |
|-----------------|--------|
| GIC ITS control | 0x0014 |

Accesses on this interface are **RW**.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_SGIR, ITS SGI Register

The GITS_SGIR characteristics are:

Purpose

Written by software to signal a virtual SGI for translation by the ITS.

Configuration

This register is present only when FEAT_GICv4p1 is implemented. Otherwise, direct accesses to GITS_SGIR are RES0.

This register is provided only in FEAT_GICv4p1 implementations.

Attributes

GITS_SGIR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|--|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | | |
| RES0 | | | | | | | | | | | | | | | | vPEID | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | vINTID | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |

Bits [63:48]

Reserved, RES0.

vPEID, bits [47:32]

ID of target vPEID.

The size of this field is IMPLEMENTATION DEFINED, and is specified by the [GICD_TYPER2.VIL](#) and [GICD_TYPER2.VID](#) fields. Unimplemented bits are RES0.

Bits [31:4]

Reserved, RES0.

vINTID, bits [3:0]

INTID of virtual SGI.

Accessing GITS_SGIR

64-bit access only.

GITS_SGIR can be accessed through the memory-mapped interfaces:

| Component | Offset |
|-----------------|---------|
| GIC ITS control | 0x20020 |

Accesses on this interface are **WO**.

GITS_STATUSR, ITS Error Reporting Status Register

The GITS_STATUSR characteristics are:

Purpose

Provides software with a mechanism to detect:

- Accesses to reserved locations.
- Writes to read-only locations.
- Reads of write-only locations.
- Unmapped MSIs.

Configuration

Attributes

GITS_STATUSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----------|----|------|------|------|-----|-----|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | Syndrome | | | | Overflow | | UMSI | WROD | RWOD | WRD | RRD | | | | | | | | | |

Bits [31:10]

Reserved, RES0.

Syndrome, bits [9:6]

Syndrome for the MSI that set GITS_STATUSR.UMSI to 1.

| Syndrome | Meaning |
|----------|------------------------|
| 0b0000 | Unknown reason. |
| 0b0010 | DeviceID out of range. |
| 0b0011 | DeviceID unmapped. |
| 0b0100 | EventID out of range. |
| 0b0101 | EventID unmapped. |
| 0b0111 | Collection unmapped. |
| 0b1001 | vPEID unmapped. |

An implementation might not support reporting all syndromes, and might report 0b0000 for any cause.

This field is UNKNOWN when GITS_STATUSR.UMSI is 0.

Overflow, bit [5]

Reports whether an unmapped MSI has been received while GITS_STATUSR.UMSI is 1.

| Overflow | Meaning |
|----------|---|
| 0b0 | No unmapped MSIs have been received since GITS_STATUSR.UMSI set to 1. |
| 0b1 | At least one unmapped MSIs have been received since GITS_STATUSR.UMSI set to 1. |

A software write of 1 to the bit clears it. A write of any other value is ignored.

If [GITS_TYPER](#).UMSI is 0, this field is RES0.

UMSI, bit [4]

Reports whether an unmapped MSI has been received

An unmapped MSI is defined as an MSI arriving at [GITS_TRANSLATER](#) for which there is insufficient mapping information for it to be forwarded to a Redistributor.

It is IMPLEMENTATION DEFINED whether an INT command can be reported as an unmapped MSI.

| UMSI | Meaning |
|------|--------------------------------------|
| 0b0 | No unmapped MSIs have been received. |
| 0b1 | Unmapped MSI received. |

A software write of 1 to the bit clears it. A write of any other value is ignored.

If [GITS_TYPER](#).UMSI is 0, this field is RES0.

WROD, bit [3]

Write to an RO location.

| WROD | Meaning |
|------|--|
| 0b0 | Normal operation. |
| 0b1 | A write to an RO location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

RWOD, bit [2]

Read of a WO location.

| RWOD | Meaning |
|------|--|
| 0b0 | Normal operation. |
| 0b1 | A read of a WO location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

WRD, bit [1]

Write to a reserved location.

| WRD | Meaning |
|-----|---|
| 0b0 | Normal operation. |
| 0b1 | A write to a reserved location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

RRD, bit [0]

Read of a reserved location.

| RRD | Meaning |
|-----|--|
| 0b0 | Normal operation. |
| 0b1 | A read of a reserved location has been detected. |

When a violation is detected, software must write 1 to this register to reset it.

Accessing GITS_STATUSR

This is an optional register. If the register is not implemented, the location is RAZ/WI.

GITS_STATUSR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------------|--------|--------------|
| GIC ITS control | 0x0040 | GITS_STATUSR |

Accesses on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_TRANSLATER, ITS Translation Register

The GITS_TRANSLATER characteristics are:

Purpose

Written by a requesting Device to signal an interrupt for translation by the ITS.

Configuration

This register is at the same offset as [GICD_SETSPI_NSR](#) in the Distributor, and is at the same offset as [GICR_SETLPIR](#) in the Redistributor.

Attributes

GITS_TRANSLATER is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EventID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

EventID, bits [31:0]

An identifier corresponding to the interrupt to be translated.

Note

The size of the EventID is DeviceID specific, and set when the DeviceID is mapped to an ITT (using MAPD).

The number of EventID bits implemented is reported by [GITS_TYPER.ID_bits](#). If a write specifies non-zero identifiers bits outside this range behavior is a CONSTRAINED UNPREDICTABLE choice between:

- Non-zero identifier bits outside the supported range are ignored.
- The write is ignored.

The DeviceID presented to an ITS is used to index a device table. The device table maps the DeviceID to an interrupt translation table for that device.

Accessing GITS_TRANSLATER

16-bit access to bits [15:0] of this register must be supported. When this register is written by a 16-bit transaction, bits [31:16] are written as zero.

Implementations must ensure that:

- A unique DeviceID is provided for each requesting device, and the DeviceID is presented to the ITS when a write to this register occurs in a manner that cannot be spoofed by any agent capable of performing writes.
- The DeviceID presented corresponds to the DeviceID field in the ITS commands.

Writes to this register are ignored if any of the following are true:

- [GITS_CTLR.Enabled](#) == 0.
- The presented DeviceID is not mapped to an Interrupt Translation Table.
- The DeviceID is larger than the supported size.
- The DeviceID is mapped to an Interrupt Translation Table, but the EventID is outside the range specified by MAPD.

- The EventID is mapped to an Interrupt Translation Table and the EventID is within the range specified by MAPD, but the EventID is unmapped.

Translation requests that result from writes to this register are subject to certain ordering rules. For more information, see 'Ordering of translations with the output to ITS commands' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

GITS_TRANSLATER can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|---------------------|--------|-----------------|
| GIC ITS translation | 0x0040 | GITS_TRANSLATER |

Accesses on this interface are **WO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_TYPER, ITS Type Register

The GITS_TYPER characteristics are:

Purpose

Specifies the features that an ITS supports.

Configuration

Attributes

GITS_TYPER is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------------|--|--|--|------|--|---------|--|---------|--|--|--|---------|--|-----|------|----------------|------|--------|------|---------------------------|------|-------|-------|-----|------|----|----|----|----|----|----|--|---|
| 636261605958575655545352 | | | | | | | | | | | | | | | | 51 | 50 | 494847 | | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | | 3 |
| RES0 | | | | | | | | | | | | | | INV | UMSI | irq | UMSI | in | DPET | VMAPP | VSGI | IMPAM | VMOVP | CIL | CIDb | | | | | | | | |
| HCC | | | | RES0 | | PTASEIS | | Devbits | | | | ID_bits | | | | ITT_entry_size | | | | IMPLEMENTATION DEFINED | | C | | | | | | | | | | | |
| 313029282726252423222120 | | | | | | | | | | | | | | | | 19 | 18 | 171615 | | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | | |

Bits [63:47]

Reserved, RES0.

INV, bit [46]

ITS cache invalidation behavior on disable.

| INV | Meaning |
|-----|---|
| 0b0 | It is IMPLEMENTATION DEFINED whether ITS caches are invalidated on clearing GITS_CTLR.Enabled and GITS_BASER<n>.Valid . |
| 0b1 | ITS caches are invalidated on clearing GITS_CTLR.Enabled and GITS_BASER<n>.Valid . |

If GITS_TYPER.INV is 1, after the following sequence:

- GITS_CTLR.Enabled written to 0.
- A read of GITS_CTLR.Quiescent returns 1.
- GITS_BASER<n>.Valid written to 0.

There is no cached information from the ITS memory structure pointed to by [GITS_BASER<n>](#).

UMSlrq, bit [45]

Indicates support for generating an interrupt on receiving unmapped MSI.

| UMSIrq | Meaning |
|--------|--|
| 0b0 | Interrupt on unmapped MSI not supported. |
| 0b1 | Interrupt on unmapped MSI is supported. |

If GITS_TYPER.UMSI is 0, this field is RES0.

UMSI, bit [44]

Indicates support for reporting receipt of unmapped MSIs.

| UMSI | Meaning |
|------|--|
| 0b0 | Reporting of unmapped MSIs is not supported. |
| 0b1 | Reporting of unmapped MSIs is supported. |

nID, bit [43]

When FEAT_GICv4p1 is implemented:

nID

| nID | Meaning |
|-----|--|
| 0b0 | Individual doorbell interrupt supported. |
| 0b1 | Individual doorbell interrupt not supported. |

Otherwise:

Reserved, RES0.

SVPET, bits [42:41]

When FEAT_GICv4p1 is implemented:

SVPET

| SVPET | Meaning |
|-------|---|
| 0b00 | vPE Table is not shared with Redistributors. |
| 0b01 | vPE Table is shared with the groups of Redistributors indicated by GITS_MPIDR.Aff3. |
| 0b10 | vPE Table is shared with the groups of Redistributors indicated by GITS_MPIDR fields Aff3 and Aff2. |
| 0b11 | vPE Table is shared with the groups of Redistributors indicated by GITS_MPIDR fields Aff3, Aff2 and Aff1. |

Otherwise:

Reserved, RES0.

VMAPP, bit [40]

When FEAT_GICv4p1 is implemented:

VMAPP

| VMAPP | Meaning |
|-------|------------------------------------|
| 0b0 | FEAT_GICv4 VMAPP command layout. |
| 0b1 | FEAT_GICv4p1 VMAPP command layout. |

Otherwise:

Reserved, RES0.

VSGI, bit [39]

When FEAT_GICv4p1 is implemented:

VSGI

| VSGI | Meaning |
|------|--|
| 0b0 | Direct injection of SGIs is not supported. |
| 0b1 | Direct injection of SGIs is supported. |

Otherwise:

Reserved, RES0.

MPAM, bit [38]

When FEAT_GICv3p1 is implemented:

MPAM

| MPAM | Meaning |
|------|------------------------|
| 0b0 | MPAM is not supported. |
| 0b1 | MPAM is supported. |

Otherwise:

Reserved, RES0.

VMOVP, bit [37]

Indicates the form of the VMOVP command.

| VMOVP | Meaning |
|-------|---|
| 0b0 | When moving a vPE, software must issue a VMOVP on all ITSs that have mappings for that vPE. The ITSList and Sequence Number fields in the VMOVP command must ensure synchronization, otherwise behavior is UNPREDICTABLE. |
| 0b1 | When moving a vPE, software must only issue a VMOVP on one of the ITSs that has a mapping for that vPE. The ITSList and Sequence Number fields in the VMOVP command are RES0. |

CIL, bit [36]

Collection ID Limit.

| CIL | Meaning |
|-----|--|
| 0b0 | ITS supports 16-bit Collection ID, GITS_TYPER.CIDbits is RES0. |
| 0b1 | GITS_TYPER.CIDbits indicates supported Collection ID size |

In implementations that do not support Collections in external memory, this bit is RES0 and the number of Collections supported is reported by [GITS_TYPER.HCC](#).

CIDbits, bits [35:32]

Number of Collection ID bits.

- The number of bits of Collection ID minus one.
- When [GITS_TYPER.CIL](#) == 0, this field is RES0.

HCC, bits [31:24]

Hardware Collection Count. The number of interrupt collections supported by the ITS without provisioning of external memory.

Note

Collections held in hardware are unmapped at reset.

Bits [23:20]

Reserved, RES0.

PTA, bit [19]

Physical Target Addresses. Indicates the format of the target address:

| PTA | Meaning |
|-----|--|
| 0b0 | The target address corresponds to the PE number specified by GICR_TYPER.Processor_Number . |
| 0b1 | The target address corresponds to the base physical address of the required Redistributor. |

For more information, see 'RDbase' in ARM® Generic Interrupt Controller Architecture Specification, GIC architecture version 3.0 and version 4.0 (ARM IHI 0069).

SEIS, bit [18]

SEI support. Indicates whether the virtual CPU interface supports generation of SEIs:

| SEIS | Meaning |
|------|--|
| 0b0 | The ITS does not support local generation of SEIs. |
| 0b1 | The ITS supports local generation of SEIs. |

Devbits, bits [17:13]

The number of DeviceID bits implemented, minus one.

ID_bits, bits [12:8]

The number of EventID bits implemented, minus one.

ITT_entry_size, bits [7:4]

Read-only. Indicates the number of bytes per translation table entry, minus one.

For more information about the ITS command 'MAPD', see MAPD.

IMPLEMENTATION DEFINED, bit [3]

IMPLEMENTATION DEFINED.

CCT, bit [2]

Cumulative Collection Tables.

| CCT | Meaning |
|-----|--|
| 0b0 | The total number of supported collections is determined by the number of collections held in memory only. |
| 0b1 | The total number of supported collections is determined by number of collections that are held in memory and the number indicated by GITS_TYPER.HCC. |

If GITS_TYPER.HCC == 0, or if memory backed collections are not supported (all [GITS_BASER<n>.Type](#) != 100), this bit is RES0.

Virtual, bit [1]**When FEAT_GICv4 is implemented:**

Indicates whether the ITS supports virtual LPIs and direct injection of virtual LPIs:

| Virtual | Meaning |
|---------|--|
| 0b0 | The ITS does not support virtual LPIs or direct injection of virtual LPIs. |
| 0b1 | The ITS supports virtual LPIs and direct injection of virtual LPIs. |

Otherwise:

Reserved, RES0.

Physical, bit [0]

Indicates whether the ITS supports physical LPIs:

| Physical | Meaning |
|----------|---|
| 0b0 | The ITS does not support physical LPIs. |
| 0b1 | The ITS supports physical LPIs. |

This field is RES1, indicating that the ITS supports physical LPIs.

Accessing GITS_TYPER**GITS_TYPER can be accessed through the memory-mapped interfaces:**

| Component | Offset | Instance |
|-----------------|--------|------------|
| GIC ITS control | 0x0008 | GITS_TYPER |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

GITS_UMSIR, ITS Unmapped MSI register

The GITS_UMSIR characteristics are:

Purpose

Provides the DeviceID and EventID of the unmapped MSI that set [GITS_STATUSR](#).UMSI.

Configuration

This register is present only when GITS_TYPER.UMSI == 1. Otherwise, direct accesses to GITS_UMSIR are RES0.

Attributes

GITS_UMSIR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| DeviceID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| EventID | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

DeviceID, bits [63:32]

DeviceID of MSI that set [GITS_STATUSR](#).UMSI to 1.

If [GITS_STATUSR](#).UMSI is 0, this field is UNKNOWN.

EventID, bits [31:0]

EventID of MSI that set [GITS_STATUSR](#).UMSI to 1.

If [GITS_STATUSR](#).UMSI is 0, this field is UNKNOWN.

Accessing GITS_UMSIR

GITS_UMSIR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------------|--------|------------|
| GIC ITS control | 0x0048 | GITS_UMSIR |

Accesses on this interface are **RO**.

MIDR_EL1, Main ID Register

The MIDR_EL1 characteristics are:

Purpose

Provides identification information for the PE, including an implementer code for the device and a device ID number.

Configuration

External register MIDR_EL1 bits [31:0] are architecturally mapped to AArch64 System register [MIDR_EL1\[31:0\]](#).

External register MIDR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [MIDR\[31:0\]](#).

It is IMPLEMENTATION DEFINED whether MIDR_EL1 is implemented in the Core power domain or in the Debug power domain.

Attributes

MIDR_EL1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|----|----|----|----|----|----|----|---------|----|----|----|--------------|----|----|----|---------|----|----|----|----|----|---|---|----------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Implementer | | | | | | | | Variant | | | | Architecture | | | | PartNum | | | | | | | | Revision | | | | | | | |

Implementer, bits [31:24]

The Implementer code. This field must hold an implementer code that has been assigned by Arm. Assigned codes include the following:

| Implementer | Meaning |
|-------------|--|
| 0x00 | Reserved for software use. |
| 0x41 | Arm Limited. |
| 0x42 | Broadcom Corporation. |
| 0x43 | Cavium Inc. |
| 0x44 | Digital Equipment Corporation. |
| 0x46 | Fujitsu Ltd. |
| 0x49 | Infineon Technologies AG. |
| 0x4D | Motorola or Freescale Semiconductor Inc. |
| 0x4E | NVIDIA Corporation. |
| 0x50 | Applied Micro Circuits Corporation. |
| 0x51 | Qualcomm Inc. |
| 0x56 | Marvell International Ltd. |
| 0x69 | Intel Corporation. |
| 0xC0 | Ampere Computing. |

Arm can assign codes that are not published in this manual. All values not assigned by Arm are reserved and must not be used.

Access to this field is **RO**.

Variant, bits [23:20]

Variant number. Typically, this field is used to distinguish between different product variants, or major revisions of a product.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Architecture, bits [19:16]

Architecture version. Defined values are:

| Architecture | Meaning |
|--------------|---|
| 0b0001 | Armv4. |
| 0b0010 | Armv4T. |
| 0b0011 | Armv5 (obsolete). |
| 0b0100 | Armv5T. |
| 0b0101 | Armv5TE. |
| 0b0110 | Armv5TEJ. |
| 0b0111 | Armv6. |
| 0b1111 | Architectural features are individually identified in the ID * registers. |

All other values are reserved.

Access to this field is **RO**.

PartNum, bits [15:4]

Primary Part Number for the device.

On processors implemented by Arm, if the top four bits of the primary part number are 0x0 or 0x7, the variant and architecture are encoded differently.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [3:0]

Revision number for the device.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing MIDR_EL1

MIDR_EL1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| Debug | 0xD00 | MIDR_EL1 |

This interface is accessible as follows:

- When IsCorePowered() and !DoubleLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register are **IMPDEF**.

MPAMCFG_CASSOC, MPAM Cache Maximum Associativity Partition Configuration Register

The MPAMCFG_CASSOC characteristics are:

Purpose

The MPAMCFG_CASSOC is a 32-bit read/write register that controls the maximum fraction of the cache associativity that the PARTID selected by [MPAMCFG_PART_SEL](#) is permitted to allocate.

MPAMCFG_CASSOC_s controls the cache maximum associativity for the Secure PARTID selected by the Secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_CASSOC_ns controls the cache maximum associativity for the Non-secure PARTID selected by the Non-secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_CASSOC_rl controls the cache maximum associativity for the Realm PARTID selected by the Realm instance of [MPAMCFG_PART_SEL](#). MPAMCFG_CASSOC_rt controls the cache maximum associativity for the Root PARTID selected by the Root instance of [MPAMCFG_PART_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the control settings accessed are those of the resource instance currently selected by [MPAMCFG_PART_SEL.RIS](#) and the PARTID selected by [MPAMCFG_PART_SEL.PARTID_SEL](#).

Configuration

The power domain of MPAMCFG_CASSOC is IMPLEMENTATION DEFINED.

This register is present only when [MPAMF_IDR.HAS_CCAP_PART](#) == 1, (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and [MPAMF_CCAP_IDR.HAS_CASSOC](#) == 1. Otherwise, direct accesses to MPAMCFG_CASSOC are RES0.

Attributes

MPAMCFG_CASSOC is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | CASSOC | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

CASSOC, bits [15:0]

Maximum cache associativity usage in fixed-point fraction format by the partition selected by [MPAMCFG_PART_SEL](#). The fraction represents the portion of the cache associativity that the PARTID is permitted to allocate. CASSOC controls the fraction of associativity in each associativity grouping of the cache. In a set associative cache, CASSOC applies to the fraction of the ways in each set.

The implemented width of the fixed-point fraction is given in [MPAMF_CCAP_IDR.CASSOC_WD](#). Unimplemented bits within the field are RAZ/WI. The implemented bits of the CASSOC field are always the most significant bits of the field.

The fixed-point fraction CASSOC is less than 1. The implied binary point is between bits 15 and 16. This representation has as the largest fraction of the cache that can be represented in an implementation with w implemented bits is 1.0 minus one half to the power w.

Accessing MPAMCFG_CASSOC

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_CASSOC_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_CASSOC_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_CASSOC_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_CASSOC_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_CASSOC_s, MPAMCFG_CASSOC_ns, MPAMCFG_CASSOC_rt, and MPAMCFG_CASSOC_rl must be separate registers:

- The Secure instance (MPAMCFG_CASSOC_s) accesses the cache maximum associativity partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_CASSOC_ns) accesses the cache maximum associativity partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_CASSOC_rt) accesses the cache maximum associativity partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_CASSOC_rl) accesses the cache maximum associativity partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_CASSOC access the cache maximum associativity partitioning configuration settings for the cache resource instance selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_CASSOC access the cache maximum associativity partitioning configuration settings for the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_CASSOC access the cache maximum associativity partitioning configuration settings for the internal PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_CASSOC access the cache maximum associativity partitioning configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 0.

MPAMCFG_CASSOC can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|------------------|
| MPAM | MPAMF_BASE_s | 0x0118 | MPAMCFG_CASSOC_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_ns | 0x0118 | MPAMCFG_CASSOC_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_rt | 0x0118 | MPAMCFG_CASSOC_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_rl | 0x0118 | MPAMCFG_CASSOC_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MPAMCFG_CMAX, MPAM Cache Maximum Capacity Partition Configuration Register

The MPAMCFG_CMAX characteristics are:

Purpose

The MPAMCFG_CMAX is a 32-bit read/write register that controls the maximum fraction of the cache capacity that the PARTID selected by [MPAMCFG_PART_SEL](#) is permitted to allocate.

MPAMCFG_CMAX_s controls the cache maximum capacity for the Secure PARTID selected by the Secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_CMAX_ns controls the cache maximum capacity for the Non-secure PARTID selected by the Non-secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_CMAX_rt controls the cache maximum capacity for the Root PARTID selected by the Root instance of [MPAMCFG_PART_SEL](#). MPAMCFG_CMAX_rl controls the cache maximum capacity for the Realm PARTID selected by the Realm instance of [MPAMCFG_PART_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the control settings accessed are those of the resource instance currently selected by [MPAMCFG_PART_SEL.RIS](#) and the PARTID selected by [MPAMCFG_PART_SEL.PARTID_SEL](#).

Configuration

The power domain of MPAMCFG_CMAX is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_CCAP_PART == 1 and MPAMF_CCAP_IDR.NO_CMAX == 0. Otherwise, direct accesses to MPAMCFG_CMAX are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_CMAX is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SOFTLIM | | | | | | | | | | | | | | | | CMAX | | | | | | | | | | | | | | | |

SOFTLIM, bit [31]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_CCAP_IDR.HAS_CMAX_SOFTLIM == 1:

Soft limiting of CMAX. Soft limiting allows some allocations by a PARTID when its cache use is above the CMAX maximum cache capacity.

| SOFTLIM | Meaning |
|---------|--|
| 0b0 | When CMAX cache capacity is exceeded, the partition is not allowed to increase its cache capacity usage. It is only permitted to replace a line that was previously occupied by a line allocated by that PARTID. |
| 0b1 | When CMAX cache capacity is exceeded, the partition is permitted to allocate capacity beyond CMAX, but only from invalid lines or lines belonging to disabled PARTIDs. |

Otherwise:

Reserved, RES0.

Bits [30:16]

Reserved, RES0.

CMAX, bits [15:0]

Maximum cache capacity usage in fixed-point fraction format by the partition selected by [MPAMCFG_PART_SEL](#). The fraction represents the portion of the total cache capacity that the PARTID is permitted to allocate.

The implemented width of the fixed-point fraction is given in [MPAMF_CCAP_IDR](#).CMAX_WD. Unimplemented bits within the field are RAZ/WI. The implemented bits of the CMAX field are always the most significant bits of the field.

The fixed-point fraction CMAX is less than 1. The implied binary point is between bits 15 and 16. This representation has as the largest fraction of the cache that can be represented in an implementation with w implemented bits is 1.0 minus one half to the power w.

Accessing MPAMCFG_CMAX

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_CMAX_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_CMAX_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_CMAX_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_CMAX_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_CMAX_s, MPAMCFG_CMAX_ns, MPAMCFG_CMAX_rt, and MPAMCFG_CMAX_rl must be separate registers:

- The Secure instance (MPAMCFG_CMAX_s) accesses the cache capacity partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_CMAX_ns) accesses the cache capacity partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_CMAX_rt) accesses the cache capacity partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_CMAX_rl) accesses the cache capacity partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_CMAX access the cache maximum capacity partitioning configuration settings for the cache resource instance selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_CMAX access the cache maximum capacity partitioning configuration settings for the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_CMAX access the cache maximum capacity partitioning configuration settings for the internal PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_CMAX access the cache maximum capacity partitioning configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 0.

MPAMCFG_CMAX can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|----------------|
| MPAM | MPAMF_BASE_s | 0x0108 | MPAMCFG_CMAX_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------|
| MPAM | MPAMF_BASE_ns | 0x0108 | MPAMCFG_CMAX_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------|
| MPAM | MPAMF_BASE_rt | 0x0108 | MPAMCFG_CMAX_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------|
| MPAM | MPAMF_BASE_rl | 0x0108 | MPAMCFG_CMAX_rl |

When FEAT_RME is implemented access on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_CMIN, MPAM Cache Minimum Capacity Partition Configuration Register

The MPAMCFG_CMIN characteristics are:

Purpose

The MPAMCFG_CMIN is a 32-bit read/write register that controls the fraction of the cache capacity that the PARTID selected by [MPAMCFG_PART_SEL](#) has priority to allocate.

MPAMCFG_CMIN_s controls the cache minimum capacity for the Secure PARTID selected by the Secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_CMIN_ns controls the cache minimum capacity for the Non-secure PARTID selected by the Non-secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_CMIN_rl controls the cache minimum capacity for the Realm PARTID selected by the Realm instance of [MPAMCFG_PART_SEL](#). MPAMCFG_CMIN_rt controls the cache minimum capacity for the Root PARTID selected by the Root instance of [MPAMCFG_PART_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the control settings accessed are those of the resource instance currently selected by [MPAMCFG_PART_SEL.RIS](#) and the PARTID selected by [MPAMCFG_PART_SEL.PARTID_SEL](#).

Configuration

The power domain of MPAMCFG_CMIN is IMPLEMENTATION DEFINED.

This register is present only when [MPAMF_IDR.HAS_CCAP_PART](#) == 1, (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and [MPAMF_CCAP_IDR.HAS_CMIN](#) == 1. Otherwise, direct accesses to MPAMCFG_CMIN are RES0.

Attributes

MPAMCFG_CMIN is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | CMIN | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

CMIN, bits [15:0]

Minimum cache capacity usage in fixed-point fraction format by the partition selected by [MPAMCFG_PART_SEL](#). The fraction represents the portion of the total cache capacity that the PARTID has priority to allocate.

The implemented width of the fixed-point fraction is the same as the width of [MPAMCFG_CMAX](#).CMAX which is given in [MPAMF_CCAP_IDR.CMAX_WD](#). Unimplemented bits within the field are RAZ/WI. The implemented bits of the CMIN field are always the most significant bits of the field.

The fixed-point fraction CMIN is less than 1. The implied binary point is between bits 15 and 16. This representation has as the largest fraction of the cache that can be represented in an implementation with w implemented bits is 1.0 minus one half to the power w.

Accessing MPAMCFG_CMIN

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_CMIN_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_CMIN_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_CMIN_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_CMIN_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_CMIN_s, MPAMCFG_CMIN_ns, MPAMCFG_CMIN_rt, and MPAMCFG_CMIN_rl must be separate registers:

- The Secure instance (MPAMCFG_CMIN_s) accesses the cache minimum capacity partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_CMIN_ns) accesses the cache minimum capacity partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_CMIN_rt) accesses the cache minimum capacity partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_CMIN_rl) accesses the cache minimum capacity partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_CMIN access the cache minimum capacity partitioning configuration settings for the cache resource instance selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_CMIN access the cache minimum capacity partitioning configuration settings for the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_CMIN access the cache minimum capacity partitioning configuration settings for the internal PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_CMIN access the cache minimum capacity partitioning configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 0.

MPAMCFG_CMIN can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|----------------|
| MPAM | MPAMF_BASE_s | 0x0110 | MPAMCFG_CMIN_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------|
| MPAM | MPAMF_BASE_ns | 0x0110 | MPAMCFG_CMIN_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------|
| MPAM | MPAMF_BASE_rt | 0x0110 | MPAMCFG_CMIN_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------|
| MPAM | MPAMF_BASE_rl | 0x0110 | MPAMCFG_CMIN_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MPAMCFG_CPBM<n>, MPAM Cache Portion Bitmap Partition Configuration Register, n = 0 - 1023

The MPAMCFG_CPBM<n> characteristics are:

Purpose

The MPAMCFG_CPBM<n> register array gives access to the cache portion bitmap. Each register in the array is a read/write register that configures the cache portions numbered from <n * 32> to <31 + (n * 32)> that a PARTID is allowed to allocate.

After setting [MPAMCFG_PART_SEL](#) with a PARTID, software writes to the MPAMCFG_CPBM<n> register to configure which cache portions the PARTID is allowed to allocate.

The MPAMCFG_CPBM<n> register that contains the bitmap bit corresponding to cache portion p has n equal to $p[15:5]$. The field, P<x>, of that MPAMCFG_CPBM<n> register that contains the bitmap bit corresponding to cache portion p has x equal to $p[4:0]$.

MPAMCFG_CPBM<n>_s controls cache portions for the Secure PARTID selected by the Secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_CPBM<n>_ns controls the cache portions for the Non-secure PARTID selected by the Non-secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_CPBM<n>_rt controls cache portions for the Root PARTID selected by the Root instance of [MPAMCFG_PART_SEL](#). MPAMCFG_CPBM<n>_rl controls the cache portions for the Realm PARTID selected by the Non-secure instance of [MPAMCFG_PART_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the control settings accessed are those of the resource instance currently selected by [MPAMCFG_PART_SEL.RIS](#) and the PARTID selected by [MPAMCFG_PART_SEL.PARTID_SEL](#).

Configuration

The power domain of MPAMCFG_CPBM<n> is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_CPOR_PART == 1. Otherwise, direct accesses to MPAMCFG_CPBM<n> are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_CPBM<n> is a 32-bit register.

Field descriptions

| 31 | 30 | 29 | 28 | 27 | 26 | 25 |
|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|
| P<32 * n + 31> | P<32 * n + 30> | P<32 * n + 29> | P<32 * n + 28> | P<32 * n + 27> | P<32 * n + 26> | P<32 * n + 25> |

P<x + (n * 32)>, bit [x], for x = 31 to 0

Portion allocation control bit. Each cache portion allocation control bit, MPAMCFG_CPBM<n>.P<x>, grants permission to the PARTID selected by [MPAMCFG_PART_SEL](#) to allocate cache lines within cache portion <x + (n * 32)>.

| P<x + (n * 32)> | Meaning |
|------------------------------|--|
| 0b0 | The PARTID is not permitted to allocate into cache portion <x + (n * 32)>. |
| 0b1 | The PARTID is permitted to allocate within cache portion <x + (n * 32)>. |

The number of bits in the cache portion partitioning bit map of this component is given in [MPAMF_CPOR_IDR.CPBM_WD](#). CPBM_WD contains a value from 1 to 2^{15} , inclusive. Values of CPBM_WD greater than 32 require an array of 32-bit [MPAMCFG_CPBM<n>](#) registers to access the cache portion bitmap, up to 1024 registers.

Bits MPAMCFG_CPBM<n>.P<x + (n * 32)>, where <x + (n * 32)> is greater than or equal to CPBM_WD, are RES0:

- If $n > \text{MPAMF_CPOR_IDR.CPBM_WD}[15:5]$, the entire 32 P<x> are RES0.
- If $n == \text{MPAMF_CPOR_IDR.CPBM_WD}[15:5]$, bits [31: CPBM_WD[4:0]] are RES0 and the remaining bits are valid.
- If $n < \text{MPAMF_CPOR_IDR.CPBM_WD}[15:5]$, the entire 32 P<x> are valid.

Accessing MPAMCFG_CPBM<n>

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_CPBM<n>_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_CPBM<n>_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_CPBM<n>_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_CPBM<n>_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_CPBM<n>_s, MPAMCFG_CPBM<n>_ns, MPAMCFG_CPBM<n>_rt, and MPAMCFG_CPBM<n>_rl must be separate registers:

- The Secure instance (MPAMCFG_CPBM<n>_s) accesses the cache portion bitmap used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_CPBM<n>_ns) accesses the cache portion bitmap used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_CPBM<n>_rt) accesses the cache portion bitmap used for Root PARTIDs.
- The Realm instance (MPAMCFG_CPBM<n>_rl) accesses the cache portion bitmap used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_CPBM<n> access the cache portion bitmap configuration settings for the cache resource instance selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_CPBM<n> access the cache portion bitmap configuration settings for the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_CPBM<n> access the cache portion bitmap configuration settings for the internal PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_CPBM<n> access the cache portion bitmap configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 0.

MPAMCFG_CPBM<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|------------------|-------------------|
| MPAM | MPAMF_BASE_s | 0x1000 + (4 * n) | MPAMCFG_CPBM<n>_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|------------------|--------------------|
| MPAM | MPAMF_BASE_ns | 0x1000 + (4 * n) | MPAMCFG_CPBM<n>_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|------------------------|--------------------|
| MPAM | MPAMF_BASE_rt | 0x1000 + (4 * n) | MPAMCFG_CPBM<n>_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|------------------------|--------------------|
| MPAM | MPAMF_BASE_rl | 0x1000 + (4 * n) | MPAMCFG_CPBM<n>_rl |

When FEAT_RME is implemented access on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_DIS, MPAM Partition Configuration Disable Register

The MPAMCFG_DIS characteristics are:

Purpose

Disables a PARTID configuration as set in other MPAMCFG registers.

MPAMCFG_DIS_s disables a Secure PARTID. MPAMCFG_DIS_ns disables a Non-secure PARTID. MPAMCFG_DIS_rl disables a Realm PARTID. MPAMCFG_DIS_rt disables a Root PARTID.

Configuration

The power domain of MPAMCFG_DIS is IMPLEMENTATION DEFINED.

This register is present only when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_ENDIS == 1. Otherwise, direct accesses to MPAMCFG_DIS are RES0.

Attributes

MPAMCFG_DIS is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NFU | RES0 | | | | | | | | | | | | | | | PARTID | | | | | | | | | | | | | | | |

NFU, bit [31]

No Future Use.

| NFU | Meaning |
|-----|--|
| 0b0 | Control settings of the disabled PARTID must be retained. |
| 0b1 | Control settings of the disabled PARTID may take an UNKNOWN value. |

Bits [30:16]

Reserved, RES0.

PARTID, bits [15:0]

Selects the PARTID to disable.

Accessing MPAMCFG_DIS

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_DIS_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_DIS_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_DIS_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_DIS_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_DIS_s, MPAMCFG_DIS_ns, MPAMCFG_DIS_rt, and MPAMCFG_DIS_rl must be separate registers:

- The Secure instance (MPAMCFG_DIS_s) accesses the PARTID disable used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_DIS_ns) accesses the PARTID disable used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_DIS_rt) accesses the PARTID disable used for Root PARTIDs.
- The Realm instance (MPAMCFG_DIS_rl) accesses the PARTID disable used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_DIS access the PARTID disable configuration settings for the PARTID disable resource instance selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_DIS access the PARTID disable configuration settings for the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_DIS access the PARTID disable configuration settings for the internal PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_DIS access the PARTID disable configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 0.

MPAMCFG_DIS can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|---------------|
| MPAM | MPAMF_BASE_s | 0x0310 | MPAMCFG_DIS_s |

Accesses on this interface are **WO/RAZ**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------|
| MPAM | MPAMF_BASE_ns | 0x0310 | MPAMCFG_DIS_ns |

Accesses on this interface are **WO/RAZ**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------|
| MPAM | MPAMF_BASE_rt | 0x0310 | MPAMCFG_DIS_rt |

When FEAT_RME is implemented access on this interface are **WO/RAZ**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------|
| MPAM | MPAMF_BASE_rl | 0x0310 | MPAMCFG_DIS_rl |

When FEAT_RME is implemented access on this interface are **WO/RAZ**.

MPAMCFG_EN, MPAM Partition Configuration Enable Register

The MPAMCFG_EN characteristics are:

Purpose

Enables a PARTID configuration as set in other MPAMCFG registers.

MPAMCFG_EN_s enables a Secure PARTID. MPAMCFG_EN_ns enables a Non-secure PARTID. MPAMCFG_EN_rl enables a Realm PARTID. MPAMCFG_EN_rt enables a Root PARTID.

Configuration

The power domain of MPAMCFG_EN is IMPLEMENTATION DEFINED.

This register is present only when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_ENDIS == 1. Otherwise, direct accesses to MPAMCFG_EN are RES0.

Attributes

MPAMCFG_EN is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | PARTID | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

PARTID, bits [15:0]

Selects the PARTID to enable.

Accessing MPAMCFG_EN

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_EN_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_EN_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_EN_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_EN_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_EN_s, MPAMCFG_EN_ns, MPAMCFG_EN_rt, and MPAMCFG_EN_rl must be separate registers:

- The Secure instance (MPAMCFG_EN_s) accesses the PARTID enable used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_EN_ns) accesses the PARTID enable used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_EN_rt) accesses the PARTID enable used for Root PARTIDs.
- The Realm instance (MPAMCFG_EN_rl) accesses the PARTID enable used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_EN access the PARTID enable configuration settings for the PARTID enable resource instance selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_EN access the PARTID enable configuration settings for the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_EN access the PARTID enable configuration settings for the internal PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_EN access the PARTID enable configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 0.

MPAMCFG_EN can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|--------------|
| MPAM | MPAMF_BASE_s | 0x0300 | MPAMCFG_EN_s |

Accesses on this interface are **WO/RAZ**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_ns | 0x0300 | MPAMCFG_EN_ns |

Accesses on this interface are **WO/RAZ**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_rt | 0x0300 | MPAMCFG_EN_rt |

When FEAT_RME is implemented access on this interface are **WO/RAZ**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_rl | 0x0300 | MPAMCFG_EN_rl |

When FEAT_RME is implemented access on this interface are **WO/RAZ**.

MPAMCFG_EN_FLAGS, MPAM Partition Configuration Enable Flags Register

The MPAMCFG_EN_FLAGS characteristics are:

Purpose

Enable flags for 32 PARTIDs.

MPAMCFG_EN_FLAGS_s gives read/write access to 32 Secure PARTIDs. MPAMCFG_EN_FLAGS_ns gives read/write access to 32 Non-secure PARTIDs. MPAMCFG_EN_FLAGS_rl gives read/write access to 32 Realm PARTIDs. MPAMCFG_EN_FLAGS_rt gives read/write access to 32 Root PARTIDs.

Configuration

The power domain of MPAMCFG_EN_FLAGS is IMPLEMENTATION DEFINED.

This register is present only when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_ENDIS == 1. Otherwise, direct accesses to MPAMCFG_EN_FLAGS are RES0.

Attributes

MPAMCFG_EN_FLAGS is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | |
|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 |
| EN31 | EN30 | EN29 | EN28 | EN27 | EN26 | EN25 | EN24 | EN23 | EN22 | EN21 | EN20 | EN19 | EN18 | EN17 | EN16 | EN15 | EN14 | EN13 | EN12 | EN11 |

EN<x>, bit [x], for x = 31 to 0

PARTID Enable flags. The group of flags accessed is selected by [MPAMCFG_PART_SEL](#).PARTID & 0x0000001F in bit [0] to [MPAMCFG_PART_SEL](#).PARTID | 0x0000001F in bit [31].

| EN<x> | Meaning |
|-------|-------------------------|
| 0b0 | The PARTID is disabled. |
| 0b1 | The PARTID is enabled. |

Each bit in [MPAMCFG_EN_FLAGS](#) gives access to the same state as controlled by [MPAMCFG_EN](#) and [MPAMCFG_DIS](#).

Bits MPAMCFG_EN_FLAGS.EN<x>, where ([MPAMCFG_PART_SEL](#).PARTID & 0x0000001F) + x is greater than [MPAMF_IDR](#).PARTID_MAX, are not required to be implemented.

As with other partitioning controls, the enable flag for PARTID 0 must be reset to 0b1 (enabled).

Accessing MPAMCFG_EN_FLAGS

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_EN_FLAGS_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_EN_FLAGS_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_EN_FLAGS_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_EN_FLAGS_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_EN_FLAGS_s, MPAMCFG_EN_FLAGS_ns, MPAMCFG_EN_FLAGS_rt, and MPAMCFG_EN_FLAGS_rl must be separate registers:

- The Secure instance (MPAMCFG_EN_FLAGS_s) accesses the PARTID enable used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_EN_FLAGS_ns) accesses the PARTID enable used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_EN_FLAGS_rt) accesses the PARTID enable used for Root PARTIDs.
- The Realm instance (MPAMCFG_EN_FLAGS_rl) accesses the PARTID enable used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_EN_FLAGS access the PARTID enable configuration settings for the PARTID enable resource instance selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_EN_FLAGS access the PARTID enable configuration settings for the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_EN_FLAGS access the PARTID enable configuration settings for the internal PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_EN_FLAGS access the PARTID enable configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 0.

MPAMCFG_EN_FLAGS can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|--------------------|
| MPAM | MPAMF_BASE_s | 0x0320 | MPAMCFG_EN_FLAGS_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------|
| MPAM | MPAMF_BASE_ns | 0x0320 | MPAMCFG_EN_FLAGS_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------|
| MPAM | MPAMF_BASE_rt | 0x0320 | MPAMCFG_EN_FLAGS_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------|
| MPAM | MPAMF_BASE_rl | 0x0320 | MPAMCFG_EN_FLAGS_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MPAMCFG_INTPARTID, MPAM Internal PARTID Narrowing Configuration Register

The MPAMCFG_INTPARTID characteristics are:

Purpose

MPAMCFG_INTPARTID is a 32-bit read/write register that controls the mapping of the PARTID selected by [MPAMCFG_PART_SEL](#) into a narrower internal PARTID (intPARTID).

MPAMCFG_INTPARTID_s controls the mapping for the Secure PARTID selected by the Secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_INTPARTID_ns controls the mapping for the Non-secure PARTID selected by the Non-secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_INTPARTID_rt controls the mapping for the Root PARTID selected by the Root instance of [MPAMCFG_PART_SEL](#). MPAMCFG_INTPARTID_rl controls the mapping for the Realm PARTID selected by the Realm instance of [MPAMCFG_PART_SEL](#).

The MPAMCFG_INTPARTID register associates the request PARTID (reqPARTID) in the [MPAMCFG_PART_SEL](#) register with an internal PARTID (intPARTID) in this register. To set that association, store reqPARTID into the [MPAMCFG_PART_SEL](#) register and then store the intPARTID into the MPAMCFG_INTPARTID register. To read the association, store reqPARTID into the MPAMCFG_PART_SEL register and then read MPAMCFG_INTPARTID.

If the intPARTID stored into MPAMCFG_INTPARTID is out-of-range or does not have the INTERNAL bit set, the association of reqPARTID to intPARTID is not written and [MPAMF_ESR](#) is set to indicate an intPARTID_Range error.

If [MPAMCFG_PART_SEL](#).INTERNAL is 1 when MPAMCFG_INTPARTID is read or written, [MPAMF_ESR](#) is set to indicate an Unexpected_INTERNAL error.

Configuration

The power domain of MPAMCFG_INTPARTID is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_PARTID_NRW == 1. Otherwise, direct accesses to MPAMCFG_INTPARTID are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_INTPARTID is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|-----------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | INTERNAL | | INTPARTID | | | | | | | | | | | | | |

Bits [31:17]

Reserved, RES0.

INTERNAL, bit [16]

Internal PARTID flag.

This bit must be 1 when written to the register. If written as 0, the write will not update the reqPARTID to intPARTID association.

On a read of this register, the bit will always read the value last written.

INTPARTID, bits [15:0]

This field contains the intPARTID mapped to the reqPARTID in [MPAMCFG_PART_SEL](#).

The maximum intPARTID supported is [MPAMF_PARTID_NRW_IDR](#).INTPARTID_MAX.

Accessing MPAMCFG_INTPARTID

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_INTPARTID_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_INTPARTID_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_INTPARTID_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_INTPARTID_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_INTPARTID_s, MPAMCFG_INTPARTID_ns, MPAMCFG_INTPARTID_rt, and MPAMCFG_INTPARTID_rl must be separate registers:

- The Secure instance (MPAMCFG_INTPARTID_s) accesses the PARTID narrowing used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_INTPARTID_ns) accesses the PARTID narrowing used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_INTPARTID_rt) accesses the PARTID narrowing used for Root PARTIDs.
- The Realm instance (MPAMCFG_INTPARTID_rl) accesses the PARTID narrowing used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_INTPARTID access the PARTID narrowing configuration settings without being affected by [MPAMCFG_PART_SEL](#).RIS.

Loads and stores to MPAMCFG_INTPARTID access the PARTID narrowing configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 0.

MPAMCFG_INTPARTID can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|---------------------|
| MPAM | MPAMF_BASE_s | 0x0600 | MPAMCFG_INTPARTID_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_ns | 0x0600 | MPAMCFG_INTPARTID_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_rt | 0x0600 | MPAMCFG_INTPARTID_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_rl | 0x0600 | MPAMCFG_INTPARTID_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MPAMCFG_MBW_MAX, MPAM Memory Bandwidth Maximum Partition Configuration Register

The MPAMCFG_MBW_MAX characteristics are:

Purpose

MPAMCFG_MBW_MAX is a 32-bit read/write register that controls the maximum fraction of memory bandwidth that the PARTID selected by [MPAMCFG_PART_SEL](#) is permitted to use.

MPAMCFG_MBW_MAX_s controls maximum bandwidth for the Secure PARTID selected by the Secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_MBW_MAX_ns controls the maximum bandwidth for the Non-secure PARTID selected by the Non-secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_MBW_MAX_rt controls the maximum bandwidth for the Root PARTID selected by the Root instance of [MPAMCFG_PART_SEL](#). MPAMCFG_MBW_MAX_rl controls the maximum bandwidth for the Realm PARTID selected by the Realm instance of [MPAMCFG_PART_SEL](#).

A PARTID that has used more than MAX is given no access to additional bandwidth if HARDLIM == 1 or is given additional bandwidth only if there are no requests from PARTIDs that have not exceeded their MAX if HARDLIM == 0.

If [MPAMF_IDR.HAS_RIS](#) is 1, the control settings accessed are those of the resource instance currently selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

Configuration

The power domain of MPAMCFG_MBW_MAX is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_MAX == 1. Otherwise, direct accesses to MPAMCFG_MBW_MAX are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_MAX is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------|----|----------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|---------------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| HARDLIM | | RES0 | | | | | | | | | | | | | | MAX | | | | | | | | | | | | | | | |

HARDLIM, bit [31]

Hard bandwidth limiting.

| HARDLIM | Meaning |
|---------|---|
| 0b0 | When MAX bandwidth is exceeded, the partition contends with a low preference for downstream bandwidth beyond MAX. |
| 0b1 | When MAX bandwidth is exceeded, the partition does not use any more bandwidth until the memory bandwidth measurement for the partition falls below MAX. |

Bits [30:16]

Reserved, RES0.

MAX, bits [15:0]

Memory maximum bandwidth allocated to the partition selected by [MPAMCFG_PART_SEL](#). MAX is in fixed-point fraction format. The fraction represents the portion of the total memory bandwidth capacity through the controlled component that the PARTID is permitted to allocate.

The implemented width of the fixed-point fraction is given in [MPAMF_MBW_IDR.BWA_WD](#). Unimplemented bits are RAZ/WI. The implemented bits of the MAX field are always to the left of the field. For example, if BWA_WD = 3, the implemented bits are MPAMCFG_MBW_MAX[15:13] and MPAMCFG_MBW_MAX[12:0] are unimplemented.

The fixed-point fraction MAX is less than 1. The implied binary point is between bits 15 and 16. This representation has as the largest fraction of the bandwidth that can be represented in an implementation with w implemented bits is 1.0 minus one half to the power w.

Accessing MPAMCFG_MBW_MAX

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_MBW_MAX_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_MBW_MAX_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_MBW_MAX_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_MAX_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_MAX_s, MPAMCFG_MBW_MAX_ns, MPAMCFG_MBW_MAX_rt, and MPAMCFG_MBW_MAX_rl must be separate registers:

- The Secure instance (MPAMCFG_MBW_MAX_s) accesses the memory maximum bandwidth partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_MAX_ns) accesses the memory maximum bandwidth partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_MAX_rt) accesses the memory maximum bandwidth partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_MAX_rl) accesses the memory maximum bandwidth partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_MAX access the memory maximum bandwidth partitioning configuration settings for the bandwidth resource instance selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_MAX access the memory maximum bandwidth partitioning configuration settings for the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_MAX access the memory maximum bandwidth partitioning configuration settings for the internal PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_MAX access the memory maximum bandwidth partitioning configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 0.

MPAMCFG_MBW_MAX can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|-------------------|
| MPAM | MPAMF_BASE_s | 0x0208 | MPAMCFG_MBW_MAX_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_ns | 0x0208 | MPAMCFG_MBW_MAX_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|-------|--------|----------|
|-----------|-------|--------|----------|

| | | | |
|------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_rt | 0x0208 | MPAMCFG_MBW_MAX_rt |
|------|---------------|--------|--------------------|

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_rl | 0x0208 | MPAMCFG_MBW_MAX_rl |

When FEAT_RME is implemented access on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_MBW_MIN, MPAM Memory Bandwidth Minimum Partition Configuration Register

The MPAMCFG_MBW_MIN characteristics are:

Purpose

MPAMCFG_MBW_MIN is a 32-bit read/write register that controls the minimum fraction of memory bandwidth that the PARTID selected by [MPAMCFG_PART_SEL](#) is permitted to use.

MPAMCFG_MBW_MIN_s controls the minimum bandwidth for the Secure PARTID selected by the Secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_MBW_MIN_ns controls the minimum bandwidth for the Non-secure PARTID selected by the Non-secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_MBW_MIN_rt controls the minimum bandwidth for the Root PARTID selected by the Root instance of [MPAMCFG_PART_SEL](#). MPAMCFG_MBW_MIN_rl controls the minimum bandwidth for the Realm PARTID selected by the Realm instance of [MPAMCFG_PART_SEL](#).

A PARTID that has used less than MIN is given preferential access to bandwidth.

If [MPAMF_IDR.HAS_RIS](#) is 1, the control settings accessed are those of the resource instance currently selected by [MPAMCFG_PART_SEL.RIS](#) and the PARTID selected by [MPAMCFG_PART_SEL.PARTID_SEL](#).

Configuration

The power domain of MPAMCFG_MBW_MIN is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_MIN == 1. Otherwise, direct accesses to MPAMCFG_MBW_MIN are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_MIN is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | MIN | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

MIN, bits [15:0]

Memory minimum bandwidth allocated to the partition selected by [MPAMCFG_PART_SEL](#). MIN is in fixed-point fraction format. The fraction represents the portion of the total memory bandwidth capacity through the controlled component that the PARTID is permitted to allocate.

The implemented width of the fixed-point fraction is given in [MPAMF_MBW_IDR.BWA_WD](#). Unimplemented bits are RAZ/WI. The implemented bits of the MIN field are always to the left of the field. For example, if BWA_WD = 4, the implemented bits are MPAMCFG_MBW_MIN[15:12] and MPAMCFG_MBW_MIN[11:0] are unimplemented.

The fixed-point fraction MIN is less than 1. The implied binary point is between bits 15 and 16. This representation has as the largest fraction of the bandwidth that can be represented in an implementation with w implemented bits is 1.0 minus one half to the power w.

Accessing MPAMCFG_MBW_MIN

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_MBW_MIN_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_MBW_MIN_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_MBW_MIN_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_MIN_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_MIN_s, MPAMCFG_MBW_MIN_ns, MPAMCFG_MBW_MIN_rt, and MPAMCFG_MBW_MIN_rl must be separate registers:

- The Secure instance (MPAMCFG_MBW_MIN_s) accesses the memory minimum bandwidth partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_MIN_ns) accesses the memory minimum bandwidth partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_MIN_rt) accesses the memory minimum bandwidth partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_MIN_rl) accesses the memory minimum bandwidth partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_MIN access the memory minimum bandwidth partitioning configuration settings for the bandwidth resource instance selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_MIN access the memory minimum bandwidth partitioning configuration settings for the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_MIN access the memory minimum bandwidth partitioning configuration settings for the internal PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_MIN access the memory minimum bandwidth partitioning configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 0.

MPAMCFG_MBW_MIN can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|-------------------|
| MPAM | MPAMF_BASE_s | 0x0200 | MPAMCFG_MBW_MIN_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_ns | 0x0200 | MPAMCFG_MBW_MIN_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_rt | 0x0200 | MPAMCFG_MBW_MIN_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_rl | 0x0200 | MPAMCFG_MBW_MIN_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MPAMCFG_MBW_PBM<n>, MPAM Bandwidth Portion Bitmap Partition Configuration Register, n = 0 - 127

The MPAMCFG_MBW_PBM<n> characteristics are:

Purpose

The MPAMCFG_MBW_PBM<n> register array gives access to the memory bandwidth portion bitmap. Each register in the array is a read/write register that configures the bandwidth portions <32 * n> to <(32 * n) + 31> that a PARTID is allowed to allocate.

After setting [MPAMCFG_PART_SEL](#) with a PARTID, software writes to one or more of the MPAMCFG_MBW_PBM<n> registers to configure which bandwidth portions the PARTID is allowed to allocate.

The MPAMCFG_MBW_PBM<n> register that contains the bitmap bit corresponding to memory bandwidth portion p has n equal to $p[11:5]$. The field, $P<x + (32 * n)>$ of that MPAMCFG_MBW_PBM<n> register that contains the bitmap bit corresponding to memory bandwidth portion p has x equal to $p[4:0]$.

The MPAMCFG_MBW_PBM<n>_s registers control the bandwidth portion bitmap for the Secure PARTID selected by the Secure instance of [MPAMCFG_PART_SEL](#). The MPAMCFG_MBW_PBM<n>_ns registers control the bandwidth portion bitmap for the Non-secure PARTID selected by the Non-secure instance of [MPAMCFG_PART_SEL](#). The MPAMCFG_MBW_PBM<n>_rt registers control the bandwidth portion bitmap for the Root PARTID selected by the Root instance of [MPAMCFG_PART_SEL](#). The MPAMCFG_MBW_PBM<n>_rl registers control the bandwidth portion bitmap for the Realm PARTID selected by the Realm instance of [MPAMCFG_PART_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the control settings accessed are those of the resource instance currently selected by [MPAMCFG_PART_SEL.RIS](#) and the PARTID selected by [MPAMCFG_PART_SEL.PARTID_SEL](#).

Configuration

The power domain of MPAMCFG_MBW_PBM<n> is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_PBM == 1. Otherwise, direct accesses to MPAMCFG_MBW_PBM<n> are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_PBM<n> is a 32-bit register.

Field descriptions

| 31 | 30 | 29 | 28 | 27 | 26 | 25 |
|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|--------------------------------------|
| P<32 * n + 31> | P<32 * n + 30> | P<32 * n + 29> | P<32 * n + 28> | P<32 * n + 27> | P<32 * n + 26> | P<32 * n + 25> |

P<x + (32 * n)>, bit [x], for x = 31 to 0

Portion allocation control bit. Each bandwidth portion allocation control bit MPAMCFG_MBW_PBM<n>.P<x + (32 * n)> grants permission to the PARTID selected by [MPAMCFG_PART_SEL](#) to allocate bandwidth within bandwidth portion <x + (32 * n)>.

| P<x + (32 * n)> | Meaning |
|-----------------|--|
| 0b0 | The PARTID is not permitted to allocate into bandwidth portion <x + (32 * n)>. |
| 0b1 | The PARTID is permitted to allocate within bandwidth portion <x + (32 * n)>. |

MPAMCFG_MBW_PBM<n>, MPAM Bandwidth Portion Bitmap Partition Configuration Register, n = 0 - 127

The number of bits in the bandwidth portion partitioning bit map of this component is given in [MPAMF_MBW_IDR.BWPBM_WD](#). BWPBM_WD contains a value from 1 to 2^{12} , inclusive. Values of BWPBM_WD greater than 32 require a group of 32-bit registers to access the bandwidth portion bitmap, up to 128 32-bit registers.

Bits MPAMCFG_MBW_PBM<n>.P<x + (32 * n)>>, where <x + (32 * n)> is greater than or equal to BWPBM_WD are RES0:

- If $n > \text{MPAMF_MBW_IDR.BWPBM_WD}[11:5]$, the entire 32 P<x> are RES0.
- If $n == \text{MPAMF_MBW_IDR.BWPBM_WD}[11:5]$, bits [31: BWPBM_WD[4:0]] are RES0 and the remaining bits are valid.
- If $n < \text{MPAMF_MBW_IDR.BWPBM_WD}[11:5]$, the entire 32 P<x> are valid.

Accessing MPAMCFG_MBW_PBM<n>

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_MBW_PBM<n>_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_MBW_PBM<n>_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_MBW_PBM<n>_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_PBM<n>_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_PBM<n>_s, MPAMCFG_MBW_PBM<n>_ns, MPAMCFG_MBW_PBM<n>_rt, and MPAMCFG_MBW_PBM<n>_rl must be separate registers:

- The Secure instance (MPAMCFG_MBW_PBM<n>_s) accesses the memory bandwidth portion bitmap used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_PBM<n>_ns) accesses the memory bandwidth portion bitmap used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_PBM<n>_rt) accesses the memory bandwidth portion bitmap used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_PBM<n>_rl) accesses the memory bandwidth portion bitmap used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_PBM<n> access the memory bandwidth portion bitmap configuration settings for the bandwidth resource instance selected by [MPAMCFG_PART_SEL.RIS](#) and the PARTID selected by [MPAMCFG_PART_SEL.PARTID_SEL](#).

When RIS is not implemented, loads and stores to MPAMCFG_MBW_PBM<n> access the memory bandwidth portion bitmap configuration settings for the PARTID selected by [MPAMCFG_PART_SEL.PARTID_SEL](#).

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_PBM<n> access the memory bandwidth portion bitmap configuration settings for the internal PARTID selected by [MPAMCFG_PART_SEL.PARTID_SEL](#), and [MPAMCFG_PART_SEL.INTERNAL](#) must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_PBM<n> access the memory bandwidth portion bitmap configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL.PARTID_SEL](#), and [MPAMCFG_PART_SEL.INTERNAL](#) must be 0.

MPAMCFG_MBW_PBM<n> can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|------------------|----------------------|
| MPAM | MPAMF_BASE_s | 0x2000 + (4 * n) | MPAMCFG_MBW_PBM<n>_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|------------------|-----------------------|
| MPAM | MPAMF_BASE_ns | 0x2000 + (4 * n) | MPAMCFG_MBW_PBM<n>_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|------------------------|-----------------------|
| MPAM | MPAMF_BASE_rt | 0x2000 + (4 * n) | MPAMCFG_MBW_PBM<n>_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|------------------------|-----------------------|
| MPAM | MPAMF_BASE_rl | 0x2000 + (4 * n) | MPAMCFG_MBW_PBM<n>_rl |

When FEAT_RME is implemented access on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_MBW_PROP, MPAM Memory Bandwidth Proportional Stride Partition Configuration Register

The MPAMCFG_MBW_PROP characteristics are:

Purpose

Controls the proportional stride of memory bandwidth that the PARTID selected by [MPAMCFG_PART_SEL](#) uses.

MPAMCFG_MBW_PROP_s controls the bandwidth proportional stride for the Secure PARTID selected by the Secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_MBW_PROP_ns controls the bandwidth proportional stride for the Non-secure PARTID selected by the Non-secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_MBW_PROP_rt controls the bandwidth proportional stride for the Root PARTID selected by the Root instance of [MPAMCFG_PART_SEL](#). MPAMCFG_MBW_PROP_rl controls the bandwidth proportional stride for the Realm PARTID selected by the Realm instance of [MPAMCFG_PART_SEL](#).

Proportional stride is a relative cost of bandwidth requested by one PARTID in relation to the costs of the bandwidths requested by each other PARTID also competing to use the bandwidth.

If [MPAMF_IDR.HAS_RIS](#) is 1, the control settings accessed are those of the resource instance currently selected by [MPAMCFG_PART_SEL.RIS](#) and the PARTID selected by [MPAMCFG_PART_SEL.PARTID_SEL](#).

Configuration

The power domain of MPAMCFG_MBW_PROP is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MBW_PART == 1 and MPAMF_MBW_IDR.HAS_PROP == 1. Otherwise, direct accesses to MPAMCFG_MBW_PROP are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_PROP is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------|----------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------------------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| EN | RES0 | | | | | | | | | | | | | | | STRIDEM1 | | | | | | | | | | | | | | | |

EN, bit [31]

Enable proportional stride bandwidth partitioning.

| EN | Meaning |
|-----|---|
| 0b0 | The selected partition is not regulated by proportional stride bandwidth partitioning. |
| 0b1 | The selected partition has bandwidth usage regulated by proportional stride bandwidth partitioning as controlled by STRIDEM1. |

Bits [30:16]

Reserved, RES0.

STRIDEM1, bits [15:0]

Memory bandwidth stride minus 1 allocated to the partition selected by [MPAMCFG_PART_SEL](#). STRIDEM1 represents the normalized cost of bandwidth consumption by the partition.

The proportional stride partitioning control parameter is an unsigned integer representing the normalized cost to a partition for consuming bandwidth. Larger values have a larger cost and correspond to a lesser allocation of bandwidth while smaller values indicate a lesser cost and therefore a higher allocation of bandwidth.

The implemented width of STRIDEM1 is given in MPAMF_MBW_IDR.BWA_WD.

Accessing MPAMCFG_MBW_PROP

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_MBW_PROP_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_MBW_PROP_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_MBW_PROP_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_PROP_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_PROP_s, MPAMCFG_MBW_PROP_ns, MPAMCFG_MBW_PROP_rt, and MPAMCFG_MBW_PROP_rl must be separate registers:

- The Secure instance (MPAMCFG_MBW_PROP_s) accesses the memory proportional stride bandwidth partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_PROP_ns) accesses the memory proportional stride bandwidth partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_PROP_rt) accesses the memory proportional stride bandwidth partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_PROP_rl) accesses the memory proportional stride bandwidth partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_PROP access the memory proportional stride bandwidth partitioning configuration settings for the bandwidth resource instance selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_PROP access the memory proportional stride bandwidth partitioning configuration settings for the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_PROP access the memory proportional stride bandwidth partitioning configuration settings for the internal PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_PROP access the memory proportional stride bandwidth partitioning configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 0.

MPAMCFG_MBW_PROP can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|--------------------|
| MPAM | MPAMF_BASE_s | 0x0500 | MPAMCFG_MBW_PROP_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------|
| MPAM | MPAMF_BASE_ns | 0x0500 | MPAMCFG_MBW_PROP_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------|
| MPAM | MPAMF_BASE_rt | 0x0500 | MPAMCFG_MBW_PROP_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------|
| MPAM | MPAMF_BASE_r1 | 0x0500 | MPAMCFG_MBW_PROP_r1 |

When FEAT_RME is implemented access on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_MBW_WINWD, MPAM Memory Bandwidth Partitioning Window Width Configuration Register

The MPAMCFG_MBW_WINWD characteristics are:

Purpose

MPAMCFG_MBW_WINWD is a 32-bit register that shows and sets the value of the window width for the PARTID in [MPAMCFG_PART_SEL](#).

MPAMCFG_MBW_WINWD_s reads and controls the bandwidth control window width for the Secure PARTID selected by the Secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_MBW_WINWD_ns reads and controls the bandwidth control window width for the Non-secure PARTID selected by the Non-secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_MBW_WINWD_rt reads and controls the bandwidth control window width for the Root PARTID selected by the Root instance of [MPAMCFG_PART_SEL](#). MPAMCFG_MBW_WINWD_rl reads and controls the bandwidth control window width for the Real PARTID selected by the Realm instance of [MPAMCFG_PART_SEL](#).

MPAMCFG_MBW_WINWD is read-only if [MPAMF_MBW_IDR](#).WINDWR == 0, and the window width is set by the hardware, even if variable.

MPAMCFG_MBW_WINWD is read/write if [MPAMF_MBW_IDR](#).WINDWR == 1, permitting configuration of the window width for each PARTID independently on hardware that supports this functionality.

If [MPAMF_IDR](#).HAS_RIS is 1, the control settings accessed are those of the resource instance currently selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

Configuration

The power domain of MPAMCFG_MBW_WINWD is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_MBW_PART == 1. Otherwise, direct accesses to MPAMCFG_MBW_WINWD are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_MBW_WINWD is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|--------|----|----|----|----|----|----|----|---------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | US INT | | | | | | | | US FRAC | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

US_INT, bits [23:8]

Window width, integer microseconds.

This field reads (and sets) the integer part of the window width in microseconds for the PARTID selected by [MPAMCFG_PART_SEL](#).

US_FRAC, bits [7:0]

Window width, fractional microseconds.

This field reads (and sets) the fractional part of the window width in microseconds for the PARTID selected by [MPAMCFG_PART_SEL](#).

Accessing MPAMCFG_MBW_WINWD

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_MBW_WINWD_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_MBW_WINWD_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_MBW_WINWD_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_MBW_WINWD_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_MBW_WINWD_s, MPAMCFG_MBW_WINWD_ns, MPAMCFG_MBW_WINWD_rt, and MPAMCFG_MBW_WINWD_rl must be separate registers:

- The Secure instance (MPAMCFG_MBW_WINWD_s) accesses the window width used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_MBW_WINWD_ns) accesses the window width used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_MBW_WINWD_rt) accesses the window width used for Root PARTIDs.
- The Realm instance (MPAMCFG_MBW_WINWD_rl) accesses the window width used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_MBW_WINWD access the window width configuration settings for the bandwidth resource instance selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_MBW_WINWD access the window width configuration settings for the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_MBW_WINWD access the window width configuration settings for the internal PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_MBW_WINWD access the window width configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 0.

MPAMCFG_MBW_WINWD can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|---------------------|
| MPAM | MPAMF_BASE_s | 0x0220 | MPAMCFG_MBW_WINWD_s |

This interface is accessible as follows:

- When MPAMF_MBW_IDR.WINDWR == 0 accesses to this register are **RO**.
- When MPAMF_MBW_IDR.WINDWR == 1 accesses to this register are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_ns | 0x0220 | MPAMCFG_MBW_WINWD_ns |

This interface is accessible as follows:

- When MPAMF_MBW_IDR.WINDWR == 0 accesses to this register are **RO**.
- When MPAMF_MBW_IDR.WINDWR == 1 accesses to this register are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_rt | 0x0220 | MPAMCFG_MBW_WINWD_rt |

This interface is accessible as follows:

- When FEAT_RME is implemented and MPAMF_MBW_IDR.WINDWR == 0 accesses to this register are **RO**.
- When FEAT_RME is implemented and MPAMF_MBW_IDR.WINDWR == 1 accesses to this register are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_rl | 0x0220 | MPAMCFG_MBW_WINWD_rl |

This interface is accessible as follows:

- When FEAT_RME is implemented and MPAMF_MBW_IDR.WINDWR == 0 accesses to this register are **RO**.
- When FEAT_RME is implemented and MPAMF_MBW_IDR.WINDWR == 1 accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_PART_SEL, MPAM Partition Configuration Selection Register

The MPAMCFG_PART_SEL characteristics are:

Purpose

Selects a partition ID to configure.

MPAMCFG_PART_SEL_s selects a Secure PARTID to configure. MPAMCFG_PART_SEL_ns selects a Non-secure PARTID to configure. MPAMCFG_PART_SEL_rt selects a Root PARTID to configure. MPAMCFG_PART_SEL_rl selects a Realm PARTID to configure.

After setting this register with a PARTID, software (usually a hypervisor) can perform a series of accesses to MPAMCFG registers to configure parameters for MPAM resource controls to use when requests have that PARTID.

Configuration

The power domain of MPAMCFG_PART_SEL is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and (MPAMF_IDR.HAS_CCAP_PART == 1, or MPAMF_IDR.HAS_CPOR_PART == 1, or MPAMF_IDR.HAS_MBW_PART == 1, or MPAMF_IDR.HAS_PRI_PART == 1, or MPAMF_IDR.HAS_PARTID_NRW == 1, or (MPAMF_IDR.EXT == 0 and MPAMF_IDR.HAS_IMPL_IDR == 1) or (MPAMF_IDR.EXT == 1, MPAMF_IDR.HAS_IMPL_IDR == 1 and MPAMF_IDR.NO_IMPL_PART == 0)). Otherwise, direct accesses to MPAMCFG_PART_SEL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_PART_SEL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-----|----|----|----|------|----|----|----|----------|----|----|----|------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | RIS | | | | RES0 | | | | INTERNAL | | | | PARTID_SEL | | | | | | | | | | | | | | | |

Bits [31:28]

Reserved, RES0.

RIS, bits [27:24]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented), MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_RIS == 1:

Resource Instance Selector. RIS selects one resource to configure through MPAMCFG registers and describe with MPAMF ID registers.

Otherwise:

Reserved, RES0.

Bits [23:17]

Reserved, RES0.

INTERNAL, bit [16]

Internal PARTID.

If [MPAMF_IDR.HAS_PARTID_NRW](#) = 0, this field is RAZ/WI.

If [MPAMF_IDR.HAS_PARTID_NRW](#) = 1:

| INTERNAL | Meaning |
|----------|--|
| 0b0 | PARTID_SEL is interpreted as a request PARTID and ignored except for use with MPAMCFG_INTPARTID register access. |
| 0b1 | PARTID_SEL is interpreted as an internal PARTID and used for access to MPAMCFG control settings except for MPAMCFG_INTPARTID . |

If PARTID narrowing is implemented as indicated by [MPAMF_IDR.HAS_PARTID_NRW](#) = 1, when accessing other MPAMCFG registers the value of the MPAMCFG_PART_SEL.INTERNAL bit is checked for these conditions:

- When the [MPAMCFG_INTPARTID](#) register is read or written, if the value of MPAMCFG_PART_SEL.INTERNAL is not 0, an Unexpected_INTERNAL error is set in [MPAMF_ESR](#).
- When an MPAMCFG register other than [MPAMCFG_INTPARTID](#) is read or written, if the value of MPAMCFG_PART_SEL.INTERNAL is not 1, [MPAMF_ESR](#) is set to indicate an intPARTID_Range error.

In either error case listed here, the value returned by a read operation is UNPREDICTABLE, and the control settings are not affected by a write.

PARTID_SEL, bits [15:0]

Selects the partition ID to configure.

Reads and writes to other MPAMCFG registers are indexed by PARTID_SEL and by the NS bit used to access MPAMCFG_PART_SEL to access the configuration for a single partition.

Accessing MPAMCFG_PART_SEL

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MPAMCFG_PART_SEL_s must be accessible from the Secure MPAM feature page. MPAMCFG_PART_SEL_ns must be accessible from the Non-secure MPAM feature page.

MPAMCFG_PART_SEL_s and MPAMCFG_PART_SEL_ns must be separate registers. The Secure instance (MPAMCFG_PART_SEL_s) accesses the PARTID selector used for Secure PARTIDs, and the Non-secure instance (MPAMCFG_PART_SEL_ns) accesses the PARTID selector used for Non-secure PARTIDs.

MPAMCFG_PART_SEL can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|--------------------|
| MPAM | MPAMF_BASE_s | 0x0100 | MPAMCFG_PART_SEL_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------|
| MPAM | MPAMF_BASE_ns | 0x0100 | MPAMCFG_PART_SEL_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------|
| MPAM | MPAMF_BASE_rt | 0x0100 | MPAMCFG_PART_SEL_rt |

When FEAT_RME is implemented access on this interface are **RW**.

MPAMCFG_PART_SEL, MPAM Partition Configuration Selection Register

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------|
| MPAM | MPAMF_BASE_r1 | 0x0100 | MPAMCFG_PART_SEL_r1 |

When FEAT_RME is implemented access on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMCFG_PRI, MPAM Priority Partition Configuration Register

The MPAMCFG_PRI characteristics are:

Purpose

Controls the internal and downstream priority of requests attributed to the PARTID selected by [MPAMCFG_PART_SEL](#).

MPAMCFG_PRI_s controls the priorities for the Secure PARTID selected by the Secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_PRI_ns controls the priorities for the Non-secure PARTID selected by the Non-secure instance of [MPAMCFG_PART_SEL](#). MPAMCFG_PRI_rt controls the priorities for the Root PARTID selected by the Root instance of [MPAMCFG_PART_SEL](#). MPAMCFG_PRI_rl controls the priorities for the Realm PARTID selected by the Realm instance of [MPAMCFG_PART_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the control settings accessed are those of the resource instance currently selected by [MPAMCFG_PART_SEL.RIS](#) and the PARTID selected by [MPAMCFG_PART_SEL.PARTID_SEL](#).

Configuration

The power domain of MPAMCFG_PRI is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_PRI_PART == 1. Otherwise, direct accesses to MPAMCFG_PRI are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMCFG_PRI is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| DSPRI | | | | | | | | | | | | | | | | INTPRI | | | | | | | | | | | | | | | |

DSPRI, bits [31:16]

Downstream priority.

If [MPAMF_PRI_IDR.HAS_DSPRI](#) == 0, bits of this field are RES0 as this field is not used.

If [MPAMF_PRI_IDR.HAS_DSPRI](#) == 1, this field is a priority value applied to downstream communications from this MSC for transactions of the partition selected by [MPAMCFG_PART_SEL](#).

The implemented width of this field is [MPAMF_PRI_IDR.DSPRI_WD](#) bits. If the implemented width is less than the width of this field, the least significant bits are used.

The encoding of priority is 0-as-lowest or 0-as-highest priority according to the value of [MPAMF_PRI_IDR.DSPRI_0_IS_LOW](#).

INTPRI, bits [15:0]

Internal priority.

If [MPAMF_PRI_IDR.HAS_INTPRI](#) == 0, bits of this field are RES0 as this field is not used.

If [MPAMF_PRI_IDR.HAS_INTPRI](#) == 1, this field is a priority value applied internally inside this MSC for transactions of the partition selected by [MPAMCFG_PART_SEL](#).

The implemented width of this field is [MPAMF_PRI_IDR.INTPRI_WD](#) bits. If the implemented width is less than the width of this field, the least significant bits are used.

The encoding of priority is 0-as-lowest or 0-as-highest priority according to the value of [MPAMF_PRI_IDR.INTPRI_0_IS_LOW](#).

Accessing MPAMCFG_PRI

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMCFG_PRI_s must be accessible from the Secure MPAM feature page.
- MPAMCFG_PRI_ns must be accessible from the Non-secure MPAM feature page.
- MPAMCFG_PRI_rt must be accessible from the Root MPAM feature page.
- MPAMCFG_PRI_rl must be accessible from the Realm MPAM feature page.

MPAMCFG_PRI_s, MPAMCFG_PRI_ns, MPAMCFG_PRI_rt, and MPAMCFG_PRI_rl must be separate registers:

- The Secure instance (MPAMCFG_PRI_s) accesses the priority partitioning used for Secure PARTIDs.
- The Non-secure instance (MPAMCFG_PRI_ns) accesses the priority partitioning used for Non-secure PARTIDs.
- The Root instance (MPAMCFG_PRI_rt) accesses the priority partitioning used for Root PARTIDs.
- The Realm instance (MPAMCFG_PRI_rl) accesses the priority partitioning used for Realm PARTIDs.

When RIS is implemented, loads and stores to MPAMCFG_PRI access the priority partitioning configuration settings for the priority resource instance selected by [MPAMCFG_PART_SEL](#).RIS and the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When RIS is not implemented, loads and stores to MPAMCFG_PRI access the priority partitioning configuration settings for the PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL.

When PARTID narrowing is implemented, loads and stores to MPAMCFG_PRI access the priority partitioning configuration settings for the internal PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 1.

When PARTID narrowing is not implemented, loads and stores to MPAMCFG_PRI access the priority partitioning configuration settings for the request PARTID selected by [MPAMCFG_PART_SEL](#).PARTID_SEL, and [MPAMCFG_PART_SEL](#).INTERNAL must be 0.

MPAMCFG_PRI can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|---------------|
| MPAM | MPAMF_BASE_s | 0x0400 | MPAMCFG_PRI_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------|
| MPAM | MPAMF_BASE_ns | 0x0400 | MPAMCFG_PRI_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------|
| MPAM | MPAMF_BASE_rt | 0x0400 | MPAMCFG_PRI_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------|
| MPAM | MPAMF_BASE_rl | 0x0400 | MPAMCFG_PRI_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MPAMF_AIDR, MPAM Architecture Identification Register

The MPAMF_AIDR characteristics are:

Purpose

Identifies the version of the MPAM architecture that this MSC implements.

Note: The following values are defined for bits [7:0]:

- 0x01 == MPAM architecture v0.1
- 0x10 == MPAM architecture v1.0
- 0x11 == MPAM architecture v1.1

Configuration

The power domain of MPAMF_AIDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_AIDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_AIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|--------------|---|---|---|--------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | ArchMajorRev | | | | ArchMinorRev | | | |

Bits [31:8]

Reserved, RES0.

ArchMajorRev, bits [7:4]

Major revision of the MPAM architecture implemented by the MSC.

This table shows the only valid combinations of MPAM version numbers in an MSC. FORCE_NS functionality is only available in MPAM v0.1.

| ArchMajorRev | ArchMinorRev | MPAMv | Available |
|--------------|--------------|-------|--------------------------------------|
| 0 | 0 | | None. |
| 0 | 1 | v0.1 | MPAMv1.0 + MPAMv1.1 + FORCE_NS |
| 1 | 0 | v1.0 | MPAMv1.0 |
| 1 | 1 | v1.1 | MPAMv1.0 + MPAMv1.1 - FORCE_NS |

Use of MPAMv0.1 in MSCs is restricted to limited circumstances. The MSC must be able to initiate requests in the Secure address space which have MPAM PARTID forced to the Non-secure space with that forcing not controllable or observable by the software that configures the device for Secure requests. Please contact Arm before setting MPAMF_AIDR to report MPAMv0.1.

ArchMinorRev, bits [3:0]

Minor revision of the MPAM architecture implemented by the MSC.

See the table in the description of the ArchMajorRev field in this register.

Accessing MPAMF_AIDR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_AIDR is read-only.

MPAMF_AIDR must be readable from the Secure, Non-secure, Root, and Realm MPAM feature pages.

MPAMF_AIDR must have the same contents in the Secure, Non-secure, Root, and Realm MPAM feature pages.

MPAMF_AIDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|--------------|
| MPAM | MPAMF_BASE_s | 0x0020 | MPAMF_AIDR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_ns | 0x0020 | MPAMF_AIDR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_rt | 0x0020 | MPAMF_AIDR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_rl | 0x0020 | MPAMF_AIDR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

MPAMF_CCAP_IDR, MPAM Features Cache Capacity Partitioning ID register

The MPAMF_CCAP_IDR characteristics are:

Purpose

Indicates the number of fractional bits in [MPAMCFG_CMAX](#).CMAX.

MPAMF_CCAP_IDR_s indicates the number of fractional bits in the Secure instance of [MPAMCFG_CMAX](#).
MPAMF_CCAP_IDR_ns indicates the number of fractional bits in the Non-secure instance of [MPAMCFG_CMAX](#).
MPAMF_CCAP_IDR_rt indicates the number of fractional bits in the Root cache capacity control settings register field, [MPAMCFG_CMAX](#).CMAX. MPAMF_CCAP_IDR_rl indicates the number of fractional bits in the Realm cache capacity control settings register field, [MPAMCFG_CMAX](#).CMAX.

When [MPAMF_IDR](#).HAS_RIS is 1, some fields in this register give information for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS. The description of every field that is affected by [MPAMCFG_PART_SEL](#).RIS has information within the field description.

Configuration

The power domain of MPAMF_CCAP_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_CCAP_PART == 1. Otherwise, direct accesses to MPAMF_CCAP_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_CCAP_IDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------|----|----|----|---------|----|----|----|----------|----|----|----|------------|----|----|----|------|----|----|----|----|----|---|---|-----------|---|---|---|------|---|---|---|---------|--|--|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | |
| HAS CMAX_SOFTLIM | | | | NO CMAX | | | | HAS CMIN | | | | HAS CASSOC | | | | RES0 | | | | | | | | CASSOC WD | | | | RES0 | | | | CMAX WD | | | |

HAS_CMAX_SOFTLIM, bit [31]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Has soft limiting selection field in [MPAMCFG_CMAX](#).

| HAS_CMAX_SOFTLIM | Meaning |
|----------------------------------|--|
| 0b0 | If MPAMCFG_CMAX is implemented, it has no SOFTLIM field and the maximum capacity is controlled with a hard limit. |
| 0b1 | If MPAMCFG_CMAX is implemented, that register has a SOFTLIMIT field to select between hard or soft limiting to the CMAX parameter. |

If RIS is implemented, this field indicates selectable limiting for the cache maximum capacity control for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

NO_CMAX, bit [30]**When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:**

Does not have CMAX partitioning.

| NO_CMAX | Meaning |
|---------|--|
| 0b0 | MPAMCFG_CMAX is implemented. |
| 0b1 | MPAMCFG_CMAX is not implemented. |

If RIS is implemented, this field indicates the absence of a cache maximum capacity partitioning control for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

HAS_CMIN, bit [29]**When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:**

Has cache minimum capacity partitioning.

| HAS_CMIN | Meaning |
|----------|--|
| 0b0 | MPAMCFG_CMIN is not implemented. |
| 0b1 | MPAMCFG_CMIN is implemented. |

If RIS is implemented, this field indicates the presence of a cache minimum capacity partitioning control for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

HAS_CASSOC, bit [28]**When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:**

Has cache maximum associativity partitioning.

| HAS_CASSOC | Meaning |
|------------|--|
| 0b0 | MPAMCFG_CASSOC is not implemented. |
| 0b1 | MPAMCFG_CASSOC is implemented. |

If RIS is implemented, this field indicates the presence of a cache maximum associativity partitioning control for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

Bits [27:13]

Reserved, RES0.

CASSOC_WD, bits [12:8]**When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:**

Number of fractional bits implemented in the cache associativity partitioning control, [MPAMCFG_CASSOC](#).CASSOC, of this MSC. See [MPAMCFG_CASSOC](#).

If RIS is implemented, this field indicates the number of fractional bits in the cache capacity partitioning control for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

Bits [7:6]

Reserved, RES0.

CMAX_WD, bits [5:0]

Number of fractional bits implemented in the cache capacity partitioning control, [MPAMCFG_CMAX](#).CMAX, of this device. See [MPAMCFG_CMAX](#).

This field must contain a value from 1 to 16, inclusive.

If RIS is implemented, this field indicates the number of fractional bits in the cache capacity partitioning control for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Accessing MPAMF_CCAP_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_CCAP_IDR is read-only.

MPAMF_CCAP_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_CCAP_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_CCAP_IDR_s is permitted to have either the same or different contents to MPAMF_CCAP_IDR_ns, MPAMF_CCAP_IDR_rt, or MPAMF_CCAP_IDR_rl.
- MPAMF_CCAP_IDR_ns is permitted to have either the same or different contents to MPAMF_CCAP_IDR_rt or MPAMF_CCAP_IDR_rl.
- MPAMF_CCAP_IDR_rt is permitted to have either the same or different contents to MPAMF_CCAP_IDR_rl.

There must be separate registers in the Secure (MPAMF_CCAP_IDR_s), Non-secure (MPAMF_CCAP_IDR_ns), Root (MPAMF_CCAP_IDR_rt), and Realm (MPAMF_CCAP_IDR_rl) MPAM feature pages.

When [MPAMF_IDR](#).HAS_RIS is 1, MPAMF_CCAP_IDR shows the configuration of cache capacity partitioning for the cache resource instance selected by [MPAMCFG_PART_SEL](#).RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

MPAMF_CCAP_IDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|------------------|
| MPAM | MPAMF_BASE_s | 0x0038 | MPAMF_CCAP_IDR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_ns | 0x0038 | MPAMF_CCAP_IDR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_rt | 0x0038 | MPAMF_CCAP_IDR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|-------|--------|----------|
|-----------|-------|--------|----------|

| | | | |
|------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_r1 | 0x0038 | MPAMF_CCAP_IDR_r1 |
|------|---------------|--------|-------------------|

When FEAT_RME is implemented access on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_CPOR_IDR, MPAM Features Cache Portion Partitioning ID register

The MPAMF_CPOR_IDR characteristics are:

Purpose

Indicates the number of bits in [MPAMCFG_CPBM<n>](#).

MPAMF_CPOR_IDR_s indicates the number of bits in the Secure instance of [MPAMCFG_CPBM<n>](#).

MPAMF_CPOR_IDR_ns indicates the number of bits in the Non-secure instance of [MPAMCFG_CPBM<n>](#).

MPAMF_CPOR_IDR_rt indicates the number of bits in the Root instance of [MPAMCFG_CPBM<n>](#).

MPAMF_CPOR_IDR_rl indicates the number of bits in the Realm instance of [MPAMCFG_CPBM<n>](#).

When [MPAMF_IDR.HAS_RIS](#) is 1, some fields in this register give information for the resource instance selector, [MPAMCFG_PART_SEL](#).RIS. The description of every field that is affected by [MPAMCFG_PART_SEL](#).RIS has information within the field description.

Configuration

The power domain of MPAMF_CPOR_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_CPOR_PART == 1. Otherwise, direct accesses to MPAMF_CPOR_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_CPOR_IDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | CPBM_WD | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

CPBM_WD, bits [15:0]

Number of bits in the cache portion partitioning bit map of this device. See [MPAMCFG_CPBM<n>](#).

This field must contain a value from 1 to 32768, inclusive. Values greater than 32 require a group of 32-bit registers to access the CPBM, up to 1024 if CPBM_WD is the largest value.

If RIS is implemented, this field indicates the number bits in the cache portion bitmap for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Accessing MPAMF_CPOR_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_CPOR_IDR is read-only.

MPAMF_CPOR_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_CPOR_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_CPOR_IDR_s is permitted to have either the same or different contents to MPAMF_CPOR_IDR_ns, MPAMF_CPOR_IDR_rt, or MPAMF_CPOR_IDR_rl.
- MPAMF_CPOR_IDR_ns is permitted to have either the same or different contents to MPAMF_CPOR_IDR_rt or MPAMF_CPOR_IDR_rl.
- MPAMF_CPOR_IDR_rt is permitted to have either the same or different contents to MPAMF_CPOR_IDR_rl.

There must be separate registers in the Secure (MPAMF_CPOR_IDR_s), Non-secure (MPAMF_CPOR_IDR_ns), Root (MPAMF_CPOR_IDR_rt), and Realm (MPAMF_CPOR_IDR_rl) MPAM feature pages.

When [MPAMF_IDR.HAS_RIS](#) is 1, MPAMF_CPOR_IDR shows the configuration of cache portion partitioning for the cache resource instance selected by [MPAMCFG_PART_SEL.RIS](#). Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

MPAMF_CPOR_IDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|------------------|
| MPAM | MPAMF_BASE_s | 0x0030 | MPAMF_CPOR_IDR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_ns | 0x0030 | MPAMF_CPOR_IDR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_rt | 0x0030 | MPAMF_CPOR_IDR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_rl | 0x0030 | MPAMF_CPOR_IDR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

MPAMF_CSUMON_IDR, MPAM Features Cache Storage Usage Monitoring ID register

The MPAMF_CSUMON_IDR characteristics are:

Purpose

Indicates the number of cache storage usage monitor instances and other properties of the CSU monitoring.

MPAMF_CSUMON_IDR_s indicates the number and properties of Secure cache storage usage monitoring.
MPAMF_CSUMON_IDR_ns indicates the number and properties of Non-secure cache storage usage monitoring.
MPAMF_CSUMON_IDR_rt indicates the number and properties of Root cache storage usage monitoring.
MPAMF_CSUMON_IDR_rl indicates the number and properties of Realm cache storage usage monitoring.

If [MPAMF_IDR](#).HAS_RIS is 1, fields that mention RIS must reflect the properties of the resource instance currently selected by [MPAMCFG_PART_SEL](#).RIS. Fields that do not mention RIS are constant across all resource instances.

Configuration

The power domain of MPAMF_CSUMON_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MPAMF_CSUMON_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_CSUMON_IDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------|--------|---------|------|----------------|----------|----------------|------------------------------|--|--|--|--|--|--|--|--|--|-----|--|--|--|--|--|--|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 2423222120191817161514131211 | | | | | | | | | | | | | | | | | |
| HAS_CAPTURE | CSU_RO | HAS_XCL | RES0 | HAS_OFLOW_LNKG | HAS_OFSR | HAS_OFLOW_CAPT | RES0 | | | | | | | | | | NUM | | | | | | | |

HAS_CAPTURE, bit [31]

The implementation supports copying an [MSMON_CSU](#) to the corresponding [MSMON_CSU_CAPTURE](#) on a capture event.

| HAS_CAPTURE | Meaning |
|-------------|--|
| 0b0 | MSMON_CSU_CAPTURE is not implemented and there is no support for capture events in the CSU monitor. |
| 0b1 | The MSMON_CSU_CAPTURE register is implemented and the CSU monitor supports the capture event behavior. |

If RIS is implemented, this field indicates that CSU monitor capture is implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

CSU_RO, bit [30]

The implementation of [MSMON_CSU](#) is read-only.

| CSU_RO | Meaning |
|--------|--|
| 0b0 | MSMON_CSU is read/write. |
| 0b1 | MSMON_CSU is read-only. |

If RIS is implemented, this field indicates that the [MSMON_CSU](#) monitor register is read-only for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

HAS_XCL, bit [29]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Has filtering to exclude clean data and implements the [MSMON_CFG_CSU_FLT](#).XCL field.

| HAS_XCL | Meaning |
|---------|---|
| 0b0 | MSMON_CFG_CSU_FLT does not implement the XCL field. |
| 0b1 | MSMON_CFG_CSU_FLT implements the XCL field to exclude counting data in the clean state in the monitor instance. |

If RIS is implemented, this field indicates that the [MSMON_CFG_CSU_FLT](#).XCL field is implemented in the CSU monitor instances for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

Bit [28]

Reserved, RES0.

HAS_OFLOW_LNKG, bit [27]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Supports [MSMON_CFG_CSU_CTL](#).OFLOW_LNKG field to control how overflow on an instance affects other monitor instances in this MSC.

| HAS_OFLOW_LNKG | Meaning |
|----------------|--|
| 0b0 | Does not support CSU overflow linkage. |
| 0b1 | Supports CSU overflow linkage and the MSMON_CFG_CSU_CTL .OFLOW_LNKG field. |

If RIS is implemented, this field indicates that [MSMON_CFG_CSU_CTL](#).OFLOW_LNKG is implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

HAS_OFSR, bit [26]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

The CSU monitor overflow status bitmap register, [MSMON_CSU_OFSR](#), is implemented.

| HAS_OFSR | Meaning |
|----------|---|
| 0b0 | MSMON_CSU_OFSR register is not implemented. |
| 0b1 | MSMON_CSU_OFSR register is implemented. |

If RIS is implemented, this field indicates that CSU monitor overflow status bitmap register is implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

HAS_OFLOW_CAPT, bit [25]

When **FEAT_MPAMv0p1** is implemented or **FEAT_MPAMv1p1** is implemented:

Supports [MSMON_CFG_CSU_CTL](#).OFLOW_CAPT field to transfer the CSU monitor instance to its capture register on an overflow or overflow linkage event.

| HAS_OFLOW_CAPT | Meaning |
|----------------|---|
| 0b0 | Does not support capture on overflow. |
| 0b1 | Supports capture on overflow and the MSMON_CFG_CSU_CTL .OFLOW_CAPT field. |

If RIS is implemented, this field indicates that [MSMON_CFG_CSU_CTL](#).OFLOW_CAPT is implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

Bits [24:16]

Reserved, RES0.

NUM_MON, bits [15:0]

The number of cache storage usage monitor instances implemented.

The largest [MSMON_CFG_MON_SEL](#).MON_SEL value is NUM_MON minus 1.

If RIS is implemented, this field indicates the number of CSU monitor instances implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Accessing MPAMF_CSUMON_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_CSUMON_IDR is read-only.

MPAMF_CSUMON_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_CSUMON_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_CSUMON_IDR_s is permitted to have either the same or different contents to MPAMF_CSUMON_IDR_ns, MPAMF_CSUMON_IDR_rt, or MPAMF_CSUMON_IDR_rl.
- MPAMF_CSUMON_IDR_ns is permitted to have either the same or different contents to MPAMF_CSUMON_IDR_rt or MPAMF_CSUMON_IDR_rl.
- MPAMF_CSUMON_IDR_rt is permitted to have either the same or different contents to MPAMF_CSUMON_IDR_rl.

There must be separate registers in the Secure (MPAMF_CSUMON_IDR_s), Non-secure (MPAMF_CSUMON_IDR_ns), Root (MPAMF_CSUMON_IDR_rt), and Realm (MPAMF_CSUMON_IDR_rl) MPAM feature pages.

When [MPAMF_IDR](#).HAS_RIS is 1, MPAMF_CSUMON_IDR shows the configuration of cache storage usage monitoring for the cache resource instance selected by [MPAMCFG_PART_SEL](#).RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

Access to MPAMF_CSUMON_IDR is not affected by [MSMON_CFG_MON_SEL](#).RIS.

MPAMF_CSUMON_IDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|-------|--------|----------|
|-----------|-------|--------|----------|

| | | | |
|------|--------------|--------|--------------------|
| MPAM | MPAMF_BASE_s | 0x0088 | MPAMF_CSUMON_IDR_s |
|------|--------------|--------|--------------------|

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------|
| MPAM | MPAMF_BASE_ns | 0x0088 | MPAMF_CSUMON_IDR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------|
| MPAM | MPAMF_BASE_rt | 0x0088 | MPAMF_CSUMON_IDR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------|
| MPAM | MPAMF_BASE_rl | 0x0088 | MPAMF_CSUMON_IDR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_ECR, MPAM Error Control Register

The MPAMF_ECR characteristics are:

Purpose

MPAMF_ECR is a 32-bit read/write register that controls MPAM error interrupts for this MSC.

MPAMF_ECR_s controls Secure MPAM error handling. MPAMF_ECR_ns controls Non-secure MPAM error handling. MPAMF_ECR_rt controls Root MPAM error handling. MPAMF_ECR_rl controls Realm MPAM error handling.

Configuration

The power domain of MPAMF_ECR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_ECR are RES0.

If an MSC cannot encounter any of the error conditions listed in 'Errors in MSCs' in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598), both the [MPAMF_ESR](#) and MPAMF_ECR must be RAZ/WI.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ECR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|-------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | INTEN |

Bits [31:1]

Reserved, RES0.

INTEN, bit [0]

Interrupt Enable.

| INTEN | Meaning |
|-------|---|
| 0b0 | MPAM error interrupts are not signaled. |
| 0b1 | MPAM error interrupts are signaled. |

Accessing MPAMF_ECR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ECR_s must be accessible from the Secure MPAM feature page.
- MPAMF_ECR_ns must be accessible from the Non-secure MPAM feature page.
- MPAMF_ECR_rt must be accessible from the Root MPAM feature page.
- MPAMF_ECR_rl must be accessible from the Realm MPAM feature page.

MPAMF_ECR_s, MPAMF_ECR_ns, MPAMF_ECR_rt, and MPAMF_ECR_rl must be separate registers:

- The Secure instance (MPAMF_ECR_s) accesses the error interrupt controls used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ECR_ns) accesses the error interrupt controls used for Non-secure PARTIDs.
- The Root instance (MPAMF_ECR_rt) accesses the error interrupt controls used for Root PARTIDs.
- The Realm instance (MPAMF_ECR_rl) accesses the error interrupt controls used for Realm PARTIDs.

MPAMF_ECR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|-------------|
| MPAM | MPAMF_BASE_s | 0x00F0 | MPAMF_ECR_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------|
| MPAM | MPAMF_BASE_ns | 0x00F0 | MPAMF_ECR_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------|
| MPAM | MPAMF_BASE_rt | 0x00F0 | MPAMF_ECR_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------|
| MPAM | MPAMF_BASE_rl | 0x00F0 | MPAMF_ECR_rl |

When FEAT_RME is implemented access on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_ERR_MSI_ADDR_H, MPAM Error MSI High-part Address Register

The MPAMF_ERR_MSI_ADDR_H characteristics are:

Purpose

MPAMF_ERR_MSI_ADDR_H is a 32-bit read/write register for the high part of the MPAM error MSI address.

MPAMF_ERR_MSI_ADDR_H_s is the high part of the MSI write address for error interrupts related to Secure PARTIDs. MPAMF_ERR_MSI_ADDR_H_ns is the high part of the MSI write address for error interrupts related to Non-secure PARTIDs. MPAMF_ERR_MSI_ADDR_H_rt is the high part of the MSI write address for error interrupts related to Root PARTIDs. MPAMF_ERR_MSI_ADDR_H_rl is the high part of the MSI write address for error interrupts related to Realm PARTIDs.

Configuration

The power domain of MPAMF_ERR_MSI_ADDR_H is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_ADDR_H are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_ADDR_H is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | MSI_ADDR_H | | | | | | | | | | | | | | | | | | | |

Bits [31:20]

Reserved, RES0.

MSI_ADDR_H, bits [19:0]

MSI write address bits[51:32].

Accessing MPAMF_ERR_MSI_ADDR_H

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ERR_MSI_ADDR_H_s must be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_ADDR_H_ns must be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_ADDR_H_rt must be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_ADDR_H_rl must be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_ADDR_H_s, MPAMF_ERR_MSI_ADDR_H_ns, MPAMF_ERR_MSI_ADDR_H_rt, and MPAMF_ERR_MSI_ADDR_H_rl must be separate registers:

- The Secure instance (MPAMF_ERR_MSI_ADDR_H_s) accesses the high part of the memory address for MSI write to signal an MPAM error used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ERR_MSI_ADDR_H_ns) accesses the high part of the memory address for MSI write to signal an MPAM error used for Non-secure PARTIDs.
- The Root instance (MPAMF_ERR_MSI_ADDR_H_rt) accesses the high part of the memory address for MSI write to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_ADDR_H_rl) accesses the high part of the memory address for MSI write to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_ADDR_H can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|------------------------|
| MPAM | MPAMF_BASE_s | 0x00E4 | MPAMF_ERR_MSI_ADDR_H_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_ns | 0x00E4 | MPAMF_ERR_MSI_ADDR_H_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_rt | 0x00E4 | MPAMF_ERR_MSI_ADDR_H_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_rl | 0x00E4 | MPAMF_ERR_MSI_ADDR_H_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MPAMF_ERR_MSI_ADDR_L, MPAM Error MSI Low-part Address Register

The MPAMF_ERR_MSI_ADDR_L characteristics are:

Purpose

MPAMF_ERR_MSI_ADDR_L is a 32-bit read/write register for the low part of the MPAM error MSI address.

MPAMF_ERR_MSI_ADDR_L_s is the low part of the MSI write address for error interrupts related to Secure PARTIDs. MPAMF_ERR_MSI_ADDR_L_ns is the low part of the MSI write address for error interrupts related to Non-secure PARTIDs. MPAMF_ERR_MSI_ADDR_L_rt is the low part of the MSI write address for error interrupts related to Root PARTIDs. MPAMF_ERR_MSI_ADDR_L_rl is the low part of the MSI write address for error interrupts related to Realm PARTIDs.

Configuration

The power domain of MPAMF_ERR_MSI_ADDR_L is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_ADDR_L are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_ADDR_L is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| MSI_ADDR_L | | | | | | | | | | | | | | | | | | Bits[1:0] | | | | | | | | | | | | | |

MSI_ADDR_L, bits [31:2]

MSI write address bits[31:2].

Bits [1:0]

Reads as 0b00.

Access to this field is **RO**.

Accessing MPAMF_ERR_MSI_ADDR_L

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ERR_MSI_ADDR_L_s must be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_ADDR_L_ns must be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_ADDR_L_rt must be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_ADDR_L_rl must be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_ADDR_L_s, MPAMF_ERR_MSI_ADDR_L_ns, MPAMF_ERR_MSI_ADDR_L_rt, and MPAMF_ERR_MSI_ADDR_L_rl must be separate registers:

- The Secure instance (MPAMF_ERR_MSI_ADDR_L_s) accesses the low part of the memory address for MSI write to signal an MPAM error used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ERR_MSI_ADDR_L_ns) accesses the low part of the memory address for MSI write to signal an MPAM error used for Non-secure PARTIDs.
- The Root instance (MPAMF_ERR_MSI_ADDR_L_rt) accesses the low part of the memory address for MSI write to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_ADDR_L_rl) accesses the low part of the memory address for MSI write to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_ADDR_L can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|------------------------|
| MPAM | MPAMF_BASE_s | 0x00E0 | MPAMF_ERR_MSI_ADDR_L_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_ns | 0x00E0 | MPAMF_ERR_MSI_ADDR_L_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_rt | 0x00E0 | MPAMF_ERR_MSI_ADDR_L_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_rl | 0x00E0 | MPAMF_ERR_MSI_ADDR_L_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MPAMF_ERR_MSI_ATTR, MPAM Error MSI Write Attributes Register

The MPAMF_ERR_MSI_ATTR characteristics are:

Purpose

MPAMF_ERR_MSI_ATTR is a 32-bit read/write register that controls MPAM error MSI write attributes for MPAM errors in this MSC.

MPAMF_ERR_MSI_ATTR_s controls the attributes of Secure MPAM error MSI writes. MPAMF_ERR_MSI_ATTR_ns controls the attributes of Non-secure MPAM error MSI writes. MPAMF_ERR_MSI_ATTR_rt controls the attributes of Root MPAM error MSI writes. MPAMF_ERR_MSI_ATTR_rl controls the attributes of Realm MPAM error MSI writes.

Configuration

The power domain of MPAMF_ERR_MSI_ATTR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_ATTR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_ATTR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|--------|-------------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|-------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | MSI_SH | MSI_MEMATTR | RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | MSIEN |

Bits [31:30]

Reserved, RES0.

MSI_SH, bits [29:28]

Sharability attribute of MSI writes.

| MSI_SH | Meaning |
|--------|--------------------------------------|
| 0b00 | Non-shareable. |
| 0b01 | Reserved, CONSTRAINED UNPREDICTABLE. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

When MPAMF_ERR_MSI_ATTR.MSI_MEMATTR specifies a Device memory type, the contents of this field are IGNORED and Shareability is effectively Outer Shareable.

MSI_MEMATTR, bits [27:24]

Memory attributes of MSI writes.

Note: This encoding matches the VMSAv8-64 stage 2 MemAttr[3:0] field as described in the Arm ARM, except that the following encodings are Reserved (not UNPREDICTABLE) and behave as DEvice-nGnRnE: 0b0100, 0b1000, and 0b1100.

| MSI_MEMATTR | Meaning |
|-------------|--|
| 0b0000 | Device-nGnRnE. |
| 0b0001 | Device-nGnRE. |
| 0b0010 | Device-nGRE. |
| 0b0011 | Device-GRE. |
| 0b0100 | Reserved. Behave as Device-nGnRnE, 0b0000. |
| 0b0101 | Normal Inner Non-cacheable, Outer Non-cacheable. |
| 0b0110 | Normal Inner Write-Through Cacheable, Outer Non-cacheable. |
| 0b0111 | Normal Inner Write-Back Cacheable, Outer Non-cacheable. |
| 0b1000 | Reserved. Behave as Device-nGnRnE, 0b0000. |
| 0b1001 | Normal Inner Non-Cachable, Outer Write-Through Cacheable. |
| 0b1010 | Normal Inner Write-Through Cacheable, Outer Write-Through Cacheable. |
| 0b1011 | Normal Inner Write-Back Cacheable, Outer Write-Through Cacheable. |
| 0b1100 | Reserved. Behave as Device-nGnRnE, 0b0000. |
| 0b1101 | Normal Inner Non-cacheable, Outer Write-Back Cacheable. |
| 0b1110 | Normal Inner Write-Through Cacheable, Outer Write-Back Cacheable. |
| 0b1111 | Normal Inner Write-Back Cacheable, Outer Write-Back Cacheable. |

When this field specifies a Device memory type, the contents of MPAMF_ERR_MSI_ATTR.MSI_SH are IGNORED and Shareability is effectively Outer Shareable.

Device types may be implemented as any Device type with more than 'n' characters. For example, if this field is set to 0b0010, an implementation may treat the MSI write as the specified type, Device-nGRE, or as Device-nGnRE or as Device-nGnRnE.

Reserved encodings 0b0100, 0b1000, and 0b1100 must be implemented to behave the same as the 0b0000 encoding.

Bits [23:1]

Reserved, RES0.

MSIEN, bit [0]

Error interrupt MSI Enable.

| MSIEN | Meaning |
|-------|---|
| 0b0 | MPAM error MSI writes are not generated to signal enabled MPAM error interrupts. When error MSI writes are disabled, hardwired error interrupts could be generated. |
| 0b1 | MPAM error MSI writes are generated to signal enabled MPAM error interrupts. When error MSI writes are enabled, hardwired error interrupts are not generated. |

The value of this field affects whether hardwired error interrupts are generated.

The reset behavior of this field is:

- On a MSC reset, this field resets to 0.

Accessing MPAMF_ERR_MSI_ATTR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ERR_MSI_ATTR_s must be accessible from the Secure MPAM feature page.

- MPAMF_ERR_MSI_ATTR_ns must be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_ATTR_rt must be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_ATTR_rl must be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_ATTR_s, MPAMF_ERR_MSI_ATTR_ns, MPAMF_ERR_MSI_ATTR_rt, and MPAMF_ERR_MSI_ATTR_rl must be separate registers:

- The Secure instance (MPAMF_ERR_MSI_ATTR_s) accesses the memory access attributes for MSI write to signal an MPAM error used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ERR_MSI_ATTR_ns) accesses the memory access attributes for MSI write to signal an MPAM error used for Non-secure PARTIDs.
- The Root instance (MPAMF_ERR_MSI_ATTR_rt) accesses the memory access attributes for MSI write to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_ATTR_rl) accesses the memory access attributes for MSI write to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_ATTR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|----------------------|
| MPAM | MPAMF_BASE_s | 0x00EC | MPAMF_ERR_MSI_ATTR_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_ns | 0x00EC | MPAMF_ERR_MSI_ATTR_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rt | 0x00EC | MPAMF_ERR_MSI_ATTR_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rl | 0x00EC | MPAMF_ERR_MSI_ATTR_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MPAMF_ERR_MSI_DATA, MPAM Error MSI Data Register

The MPAMF_ERR_MSI_DATA characteristics are:

Purpose

MPAMF_ERR_MSI_DATA is a 32-bit read/write register for the MPAM error MSI data.

MPAMF_ERR_MSI_DATA_s is the data for the MSI write for error interrupts related to Secure PARTIDs.

MPAMF_ERR_MSI_DATA_ns is the data for the MSI write for error interrupts related to Non-secure PARTIDs.

MPAMF_ERR_MSI_DATA_rt is the data for the MSI write for error interrupts related to Root PARTIDs.

MPAMF_ERR_MSI_DATA_rl is the data for the MSI write for error interrupts related to Realm PARTIDs.

Configuration

The power domain of MPAMF_ERR_MSI_DATA is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_DATA are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_DATA is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | MSI_DATA | | | | | | | | | | | | | | | |

MSI_DATA, bits [31:0]

MSI data to be written to ITS to signal an MSI.

Accessing MPAMF_ERR_MSI_DATA

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ERR_MSI_DATA_s must be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_DATA_ns must be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_DATA_rt must be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_DATA_rl must be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_DATA_s, MPAMF_ERR_MSI_DATA_ns, MPAMF_ERR_MSI_DATA_rt, and MPAMF_ERR_MSI_DATA_rl must be separate registers:

- The Secure instance (MPAMF_ERR_MSI_DATA_s) accesses the data for MSI write to signal an MPAM error used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ERR_MSI_DATA_ns) accesses the data for MSI write to signal an MPAM error used for Non-secure PARTIDs.
- The Root instance (MPAMF_ERR_MSI_DATA_rt) accesses the data for MSI write to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_DATA_rl) accesses the data for MSI write to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_DATA can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|----------------------|
| MPAM | MPAMF_BASE_s | 0x00E8 | MPAMF_ERR_MSI_DATA_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_ns | 0x00E8 | MPAMF_ERR_MSI_DATA_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rt | 0x00E8 | MPAMF_ERR_MSI_DATA_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rl | 0x00E8 | MPAMF_ERR_MSI_DATA_rl |

When FEAT_RME is implemented access on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_ERR_MSI_MPAM, MPAM Error MSI Write MPAM Information Register

The MPAMF_ERR_MSI_MPAM characteristics are:

Purpose

MPAMF_ERR_MSI_MPAM is a 32-bit read/write register that sets the MPAM information for error MSI write attributes for MPAM errors in this MSC.

MPAMF_ERR_MSI_MPAM_s controls MPAM information labeling of Secure MPAM error MSI writes.

MPAMF_ERR_MSI_MPAM_ns controls MPAM information labeling of Non-secure MPAM error MSI writes.

MPAMF_ERR_MSI_MPAM_rt controls MPAM information labeling of Root MPAM error MSI writes.

MPAMF_ERR_MSI_MPAM_rl controls MPAM information labeling of Realm MPAM error MSI writes.

Configuration

The power domain of MPAMF_ERR_MSI_MPAM is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_IDR.HAS_ERR_MSI == 1. Otherwise, direct accesses to MPAMF_ERR_MSI_MPAM are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ERR_MSI_MPAM is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | PMG | | | | | | | | PARTID | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group property for PARTID MSC error interrupt write.

The reset behavior of this field is:

- On a MSC reset, this field resets to an architecturally UNKNOWN value.

PARTID, bits [15:0]

Partition ID for MSC error interrupt write.

The PARTID in this register is in the Secure PARTID space in the MPAMF_ERR_MSI_MPAM_s instance and in the Non-secure PARTID space in the MPAMF_ERR_MSI_MPAM_ns instance of this register.

The reset behavior of this field is:

- On a MSC reset, this field resets to an architecturally UNKNOWN value.

Accessing MPAMF_ERR_MSI_MPAM

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ERR_MSI_MPAM_s must be accessible from the Secure MPAM feature page.
- MPAMF_ERR_MSI_MPAM_ns must be accessible from the Non-secure MPAM feature page.
- MPAMF_ERR_MSI_MPAM_rt must be accessible from the Root MPAM feature page.
- MPAMF_ERR_MSI_MPAM_rl must be accessible from the Realm MPAM feature page.

MPAMF_ERR_MSI_MPAM_s, MPAMF_ERR_MSI_MPAM_ns, MPAMF_ERR_MSI_MPAM_rt, and MPAMF_ERR_MSI_MPAM_rl must be separate registers:

- The Secure instance (MPAMF_ERR_MSI_MPAM_s) accesses the MPAM information for MSI write request to signal an MPAM error used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ERR_MSI_MPAM_ns) accesses the MPAM information for MSI write request to signal an MPAM error used for Non-secure PARTIDs.
- The Root instance (MPAMF_ERR_MSI_MPAM_rt) accesses the MPAM information for MSI write request to signal an MPAM error used for Root PARTIDs.
- The Realm instance (MPAMF_ERR_MSI_MPAM_rl) accesses the MPAM information for MSI write request to signal an MPAM error used for Realm PARTIDs.

MPAMF_ERR_MSI_MPAM can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|----------------------|
| MPAM | MPAMF_BASE_s | 0x00DC | MPAMF_ERR_MSI_MPAM_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_ns | 0x00DC | MPAMF_ERR_MSI_MPAM_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rt | 0x00DC | MPAMF_ERR_MSI_MPAM_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rl | 0x00DC | MPAMF_ERR_MSI_MPAM_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MPAMF_ESR, MPAM Error Status Register

The MPAMF_ESR characteristics are:

Purpose

Indicates MPAM error status for this MSC.

MPAMF_ESR_s reports Secure MPAM errors. MPAMF_ESR_ns reports Non-secure MPAM errors. MPAMF_ESR_rt reports Root MPAM errors. MPAMF_ESR_rl reports Realm MPAM errors.

Software should write this register after reading the status of an error to reset ERRCODE to 0x0000 and OVRWR to 0 so that future errors are not reported with OVRWR set.

Configuration

The power domain of MPAMF_ESR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_ESR are RES0.

MPAMF_ESR is 64-bit register when MPAM v0.1 or v1.1 is implemented and MPAMF_IDR.HAS_EXTD_ESR == 1.

Otherwise, MPAMF_ESR is a 32-bit register.

If an MSC cannot encounter any of the error conditions listed in 'Errors in MSCs' in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598), both the MPAMF_ESR and [MPAMF_ECR](#) must be RAZ/WI.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_ESR is a:

- 64-bit register when (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_EXTD_ESR == 1
- 32-bit register otherwise

Field descriptions

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_IDR.HAS_EXTD_ESR == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|------|---------|-----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| OVRWR | RES0 | ERRCODE | PMG | | | | | | | | | | | | PARTID_MON | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:36]

Reserved, RES0.

RIS, bits [35:32]

When MPAMF_IDR.HAS_RIS == 1:

Resource Instance Selector. Where applicable to the ERRCODE, captures the RIS value for the error.

Otherwise:

Reserved, RES0.

OVRWR, bit [31]

Overwritten.

If 0 and ERRCODE == 0b0000, no errors have occurred.

If 0 and ERRCODE is non-zero, a single error has occurred and is recorded in this register.

If 1 and ERRCODE is non-zero, multiple errors have occurred and this register records the most recent error.

The state where this bit is 1 and ERRCODE is zero must not be produced by hardware and is only reached when software writes this combination into this register.

Bits [30:28]

Reserved, RES0.

ERRCODE, bits [27:24]

Error code.

| ERRCODE | Meaning |
|---------|-------------------------|
| 0b0000 | No error. |
| 0b0001 | PARTID_SEL_Range. |
| 0b0010 | Req_PARTID_Range. |
| 0b0011 | MSMONCFG_ID_RANGE. |
| 0b0100 | Req_PMG_Range. |
| 0b0101 | Monitor_Range. |
| 0b0110 | intPARTID_Range. |
| 0b0111 | Unexpected_INTERNAL. |
| 0b1000 | Undefined_RIS_PART_SEL. |
| 0b1001 | RIS_No_Control. |
| 0b1010 | Undefined_RIS_MON_SEL. |
| 0b1011 | RIS_No_Monitor. |
| 0b1100 | Reserved. |
| 0b1101 | Reserved. |
| 0b1110 | Reserved. |
| 0b1111 | Reserved. |

PMG, bits [23:16]

Program monitoring group.

Set to the PMG on an error that captures PMG. Otherwise, set to 0x00 on an error that does not capture PMG.

PARTID_MON, bits [15:0]

PARTID or monitor.

Set to the PARTID on an error that captures PARTID.

Set to the monitor index on an error that captures MON.

On an error that captures neither PARTID nor MON, this field is set to 0.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|------|---------|-----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| OVRWR | RES0 | ERRCODE | PMG | | | | | | | | | | PARTID_MON | | | | | | | | | | | | | | | | | | |

OVRWR, bit [31]

Overwritten.

If 0 and ERRCODE == 0b0000, no errors have occurred.

If 0 and ERRCODE is non-zero, a single error has occurred and is recorded in this register.

If 1 and ERRCODE is non-zero, multiple errors have occurred and this register records the most recent error.

The state where this bit is 1 and ERRCODE is 0 must not be produced by hardware and is only reached when software writes this combination into this register.

Bits [30:28]

Reserved, RES0.

ERRCODE, bits [27:24]

Error code.

| ERRCODE | Meaning |
|---------|----------------------|
| 0b0000 | No error. |
| 0b0001 | PARTID_SEL_Range. |
| 0b0010 | Req_PARTID_Range. |
| 0b0011 | MSMONCFG_ID_RANGE. |
| 0b0100 | Req_PMG_Range. |
| 0b0101 | Monitor_Range. |
| 0b0110 | intPARTID_Range. |
| 0b0111 | Unexpected_INTERNAL. |
| 0b1000 | Reserved. |
| 0b1001 | Reserved. |
| 0b1010 | Reserved. |
| 0b1011 | Reserved. |
| 0b1100 | Reserved. |
| 0b1101 | Reserved. |
| 0b1110 | Reserved. |
| 0b1111 | Reserved. |

PMG, bits [23:16]

Program monitoring group.

Set to the PMG on an error that captures PMG. Otherwise, set to 0x00 on an error that does not capture PMG.

PARTID_MON, bits [15:0]

PARTID or monitor.

Set to the PARTID on an error that captures PARTID.

Set to the monitor index on an error that captures MON.

On an error that captures neither PARTID nor MON, this field is set to 0x0000.

Accessing MPAMF_ESR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MPAMF_ESR_s must be accessible from the Secure MPAM feature page.
- MPAMF_ESR_ns must be accessible from the Non-secure MPAM feature page.
- MPAMF_ESR_rt must be accessible from the Root MPAM feature page.

- MPAMF_ESR_rl must be accessible from the Realm MPAM feature page.

MPAMF_ESR_s, MPAMF_ESR_ns, MPAMF_ESR_rt, and MPAMF_ESR_rl must be separate registers:

- The Secure instance (MPAMF_ESR_s) accesses the error status used for Secure PARTIDs.
- The Non-secure instance (MPAMF_ESR_ns) accesses the error status used for Non-secure PARTIDs.
- The Root instance (MPAMF_ESR_rt) accesses the error status used for Root PARTIDs.
- The Realm instance (MPAMF_ESR_rl) accesses the error status used for Realm PARTIDs.

MPAMF_ESR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|-------------|
| MPAM | MPAMF_BASE_s | 0x00F8 | MPAMF_ESR_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------|
| MPAM | MPAMF_BASE_ns | 0x00F8 | MPAMF_ESR_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------|
| MPAM | MPAMF_BASE_rt | 0x00F8 | MPAMF_ESR_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------|
| MPAM | MPAMF_BASE_rl | 0x00F8 | MPAMF_ESR_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MPAMF_IDR, MPAM Features Identification Register

The MPAMF_IDR characteristics are:

Purpose

Indicates which memory partitioning and monitoring features are present on this MSC.

MPAMF_IDR_s indicates the MPAM features accessed from the Secure MPAM feature page. MPAMF_IDR_ns indicates the MPAM features accessed from the Non-secure MPAM feature page. MPAMF_IDR_rt indicates the MPAM features accessed from the Root MPAM feature page. MPAMF_IDR_rl indicates the MPAM features accessed from the Realm MPAM feature page.

When MPAMF_IDR.HAS_RIS is 1, some fields in this register give information for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS. The description of every field that is affected by [MPAMCFG_PART_SEL](#).RIS has that information within the field description.

Configuration

The power domain of MPAMF_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_IDR are RES0.

MPAMF_IDR is 64-bit register when MPAM v0.1 or v1.1 is implemented.

Otherwise, MPAMF_IDR is a 32-bit register.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_IDR is a:

- 64-bit register when FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented
- 32-bit register otherwise

Field descriptions

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 |
|----------------|-----------|------------------|--------------|--------------|---------------|---------------|----|----|
| RES0 | | | | RIS_MAX | | | | |
| HAS_PARTID_NRW | HAS_MSMON | HAS_IMPL_IDR_EXT | HAS_PRI_PART | HAS_MBW_PART | HAS_CPOR_PART | HAS_CCAP_PART | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 |

Bits [63:60]

Reserved, RES0.

RIS_MAX, bits [59:56]

When MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_RIS == 1:

Maximum RIS value supported in [MPAMCFG_PART_SEL](#). Must be 0b0000 if [MPAMF_IDR](#).HAS_RIS == 0.

Otherwise:

Reserved, RES0.

Bits [55:44]

Reserved, RES0.

HAS_NFU, bit [43]

When FEAT_MPAMv1p1 is implemented or FEAT_MPAMv0p1 is implemented:

Has No Future Use field in [MPAMCFG_DIS](#). Indicates that [MPAMCFG_DIS](#).NFU is implemented.

| HAS_NFU | Meaning |
|---------|---|
| 0b0 | MPAMCFG_DIS .NFU is not implemented. A PARTID disabled through access to MPAMCFG_DIS must preserve the control settings of the disabled PARTID. |
| 0b1 | Implements MPAMCFG_DIS .NFU. A PARTID disabled with NFU as 1 may have its control settings forgotten. |

If [MPAMF_IDR](#).HAS_ENDIS is 0b0, this field must also be 0b0.

This field must be the same in each instance of this register and for any value in [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

HAS_ENDIS, bit [42]

When FEAT_MPAMv1p1 is implemented or FEAT_MPAMv0p1 is implemented:

Has PARTID enable and disable. Indicates that this MSC supports PARTID disable and enable via [MPAMCFG_DIS](#), [MPAMCFG_EN](#) and [MPAMCFG_EN_FLAGS](#) registers.

| HAS_ENDIS | Meaning |
|-----------|--|
| 0b0 | Does not support PARTID enable and disable functionality, and MPAMCFG_EN , MPAMCFG_DIS and MPAMCFG_EN_FLAGS registers are not implemented. |
| 0b1 | Supports PARTID enable and disable through the MPAMCFG_EN , MPAMCFG_DIS and MPAMCFG_EN_FLAGS registers. |

All three registers must be implemented when this field is 1, [MPAMCFG_EN](#), [MPAMCFG_DIS](#), and [MPAMCFG_EN_FLAGS](#).

This field must be the same in each instance of this register and for any value in [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

SP4, bit [41]

When FEAT_RME is implemented:

Indicates whether this MSC supports 4 PARTID spaces.

| SP4 | Meaning |
|-----|---------------------------------------|
| 0b0 | This MSC supports two PARTID spaces. |
| 0b1 | This MSC supports four PARTID spaces. |

This field must read the same in each instance of this register and for any value in [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

HAS_ERR_MSI, bit [40]When **MPAMF_IDR.EXT** == 1:

Has support for MSI writes to signal MPAM error interrupts. These registers are implemented: [MPAMF_ERR_MSI_ADDR_L](#), [MPAMF_ERR_MSI_ADDR_H](#), [MPAMF_ERR_MSI_ATTR](#), [MPAMF_ERR_MSI_DATA](#), and [MPAMF_ERR_MSI_MPAM](#).

| HAS_ERR_MSI | Meaning |
|--------------------|---|
| 0b0 | MPAMF_ERR_MSI_ADDR_L , MPAMF_ERR_MSI_ADDR_H , MPAMF_ERR_MSI_ATTR , MPAMF_ERR_MSI_DATA , and MPAMF_ERR_MSI_MPAM registers are not implemented. |
| 0b1 | MPAMF_ERR_MSI_ADDR_L , MPAMF_ERR_MSI_ADDR_H , MPAMF_ERR_MSI_ATTR , MPAMF_ERR_MSI_DATA , and MPAMF_ERR_MSI_MPAM are implemented and can be used to generate writes to signal error interrupts. |

If [MPAMF_IDR.HAS_ESR](#) is 0, this bit must also be 0.

Otherwise:

Reserved, RES0.

HAS_ESR, bit [39]When **MPAMF_IDR.EXT** == 1:

[MPAMF_ESR](#) is implemented.

| HAS_ESR | Meaning |
|----------------|--|
| 0b0 | MPAMF_ESR , MPAMF_ECR , and MPAM error handling are not implemented. |
| 0b1 | MPAMF_ESR , MPAMF_ECR , and MPAM error handling are implemented. |

If an MSC cannot encounter any of the error conditions listed in 'Errors in MSCs' in Arm® Architecture Reference Manual Supplement, Memory System Resource Partitioning and Monitoring (MPAM), for Armv8-A (ARM DDI 0598), both the [MPAMF_ESR](#) and [MPAMF_ECR](#) must be RAZ/WI.

Otherwise:

Reserved, RES0.

HAS_EXTD_ESR, bit [38]When **MPAMF_IDR.EXT** == 1:

[MPAMF_ESR](#) is 64 bits.

| HAS_EXTD_ESR | Meaning |
|---------------------|---------------------------------------|
| 0b0 | MPAMF_ESR is 32 bits. |
| 0b1 | MPAMF_ESR is 64 bits. |

When [MPAMF_IDR.HAS_RIS](#) and [MPAMF_IDR.HAS_ESR](#), this field must be 1.

Otherwise:

Reserved, RES0.

NO_IMPL_MSMON, bit [37]

When MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_IMPL_IDR == 1:

[MPAMF_IMPL_IDR](#) defines no IMPLEMENTATION DEFINED resource monitors.

| NO_IMPL_MSMON | Meaning |
|---------------|--|
| 0b0 | MPAMF_IMPL_IDR defines at least one IMPLEMENTATION DEFINED resource monitor. |
| 0b1 | MPAMF_IMPL_IDR does not define any IMPLEMENTATION DEFINED resource monitors. |

If RIS is implemented, this field indicates the presence of IMPLEMENTATION DEFINED resource monitors described in [MPAMF_IMPL_IDR](#) for the selected resource instance.

Otherwise:

Reserved, RES0.

NO_IMPL_PART, bit [36]

When MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_IMPL_IDR == 1:

[MPAMF_IMPL_IDR](#) defines no IMPLEMENTATION DEFINED resource controls.

| NO_IMPL_PART | Meaning |
|--------------|--|
| 0b0 | MPAMF_IMPL_IDR defines at least one IMPLEMENTATION DEFINED resource control. |
| 0b1 | MPAMF_IMPL_IDR does not define any IMPLEMENTATION DEFINED resource controls. |

If RIS is implemented, this field indicates the presence of IMPLEMENTATION DEFINED resource controls described in [MPAMF_IMPL_IDR](#) for the selected resource instance.

Otherwise:

Reserved, RES0.

Bits [35:33]

Reserved, RES0.

HAS_RIS, bit [32]

When MPAMF_IDR.EXT == 1:

Has resource instance selector. Indicates that [MPAMCFG_PART_SEL](#) contains the RIS field that selects a resource instance to control.

| HAS_RIS | Meaning |
|---------|---|
| 0b0 | MPAMCFG_PART_SEL does not implement the MPAMCFG_PART_SEL .RIS field or multiple resource instance support. |
| 0b1 | MPAMCFG_PART_SEL implements the MPAMCFG_PART_SEL .RIS field and MPAM resource instance numbers up to and including MPAMF_IDR.RIS_MAX. |

Otherwise:

Reserved, RES0.

HAS_PARTID_NRW, bit [31]

Has PARTID narrowing.

| HAS_PARTID_NRW | Meaning |
|-----------------------|--|
| 0b0 | Does not have MPAMF_PARTID_NRW_IDR , MPAMCFG_INTPARTID , or intPARTID mapping support. |
| 0b1 | Supports the MPAMF_PARTID_NRW_IDR , MPAMCFG_INTPARTID registers. |

HAS_MSMON, bit [30]

Has resource Monitors. Indicates whether this MSC has MPAM resource monitors.

| HAS_MSMON | Meaning |
|------------------|---|
| 0b0 | Does not support MPAM resource monitoring by groups or MPAMF_MSMON_IDR . |
| 0b1 | Supports resource monitoring by matching a combination of PARTID and PMG. See MPAMF_MSMON_IDR . |

HAS_IMPL_IDR, bit [29]

Has [MPAMF_IMPL_IDR](#). Indicates whether this MSC has the IMPLEMENTATION SPECIFIC MPAM features register, [MPAMF_IMPL_IDR](#).

| HAS_IMPL_IDR | Meaning |
|---------------------|--|
| 0b0 | Does not have MPAMF_IMPL_IDR . |
| 0b1 | Has MPAMF_IMPL_IDR . |

EXT, bit [28]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Extended MPAMF_IDR.

| EXT | Meaning |
|------------|--|
| 0b0 | MPAMF_IDR has no defined bits in [63:32]. The register is effectively 32 bits. |
| 0b1 | MPAMF_IDR has bits defined in [63:32]. The register is 64-bits. |

Otherwise:

Reserved, RES0.

HAS_PRI_PART, bit [27]

Has Priority Partitioning. Indicates that MPAM priority partitioning is implemented and [MPAMF_PRI_IDR](#) exists.

| HAS_PRI_PART | Meaning |
|---------------------|--|
| 0b0 | Does not support priority partitioning or have MPAMF_PRI_IDR . |
| 0b1 | Has priority partitioning and MPAMF_PRI_IDR . |

If RIS is implemented, this field indicates the presence of priority partitioning resource controls as described in [MPAMF_PRI_IDR](#) for the selected resource instance.

HAS_MBW_PART, bit [26]

Has Memory Bandwidth Partitioning. Indicates whether this MSC implements MPAM memory bandwidth partitioning and [MPAMF_MBW_IDR](#).

| HAS_MBW_PART | Meaning |
|--------------|--|
| 0b0 | Does not support memory bandwidth partitioning or have MPAMF_MBW_IDR register. |
| 0b1 | Has MPAMF_MBW_IDR register. |

If RIS is implemented, this field indicates the presence of memory bandwidth partitioning resource controls as described in [MPAMF_MBW_IDR](#) for the selected resource instance.

HAS_CPOR_PART, bit [25]

Has Cache Portion Partitioning. Indicates whether this MSC implements MPAM cache portion partitioning and [MPAMF_CPOR_IDR](#).

| HAS_CPOR_PART | Meaning |
|---------------|---|
| 0b0 | Does not support cache portion partitioning or have MPAMF_CPOR_IDR or MPAMCFG_CPB<n> registers. |
| 0b1 | Has MPAMF_CPOR_IDR and MPAMCFG_CPB<n> registers. |

If RIS is implemented, this field indicates the presence of cache portion partitioning resource controls as described in [MPAMF_CPOR_IDR](#) for the selected resource instance.

HAS_CCAP_PART, bit [24]

Has Cache Capacity Partitioning. Indicates whether this MSC implements MPAM cache capacity partitioning and the [MPAMF_CCAP_IDR](#) and [MPAMCFG_CMAX](#) registers.

| HAS_CCAP_PART | Meaning |
|---------------|---|
| 0b0 | Does not support cache capacity partitioning or have MPAMF_CCAP_IDR and MPAMCFG_CMAX registers. |
| 0b1 | Has MPAMF_CCAP_IDR and MPAMCFG_CMAX registers. |

If RIS is implemented, this field indicates the presence of cache capacity partitioning resource controls as described in [MPAMF_CPOR_IDR](#) for the selected resource instance.

PMG_MAX, bits [23:16]

Maximum supported value of PMG.

The value of this field is permitted to vary between the instances of [MPAM_IDR](#), each reporting the maximum supported PMG value in the PARTID space associated with that instance.

In [MPAMF_IDR_s](#), this field is permitted to report the maximum PMG value for the Non-secure PARTID space or for the Secure PARTID space. The maximum PMG value for the Secure PARTID space can be read from [MPAMF_SIDR.PMG_MAX](#).

PARTID_MAX, bits [15:0]

Maximum supported value of PARTID.

The value of this field is permitted to vary between the instances of [MPAM_IDR](#), each reporting the maximum supported PARTID value in the PARTID space associated with that instance.

In [MPAMF_IDR_s](#), this field is permitted to report the maximum PARTID value for the Non-secure PARTID space or for the Secure PARTID space. The maximum PARTID value for the Secure PARTID space can be read from [MPAMF_SIDR.PARTID_MAX](#).

Otherwise:

| | | | | | | | | |
|--------------------------------|---------------------------|------------------------------|---------------------|------------------------------|------------------------------|-------------------------------|-------------------------------|--------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 |
| HAS_PARTID_NRW | HAS_MSMON | HAS_IMPL_IDR | EXT | HAS_PRI_PART | HAS_MBW_PART | HAS_CPOR_PART | HAS_CCAP_PART | HAS_CMAX |

HAS_PARTID_NRW, bit [31]

Has PARTID Narrowing.

| HAS_PARTID_NRW | Meaning |
|-----------------------|--|
| 0b0 | Does not have MPAMF_PARTID_NRW_IDR , MPAMCFG_INTPARTID , or intPARTID mapping support. |
| 0b1 | Supports the MPAMF_PARTID_NRW_IDR , MPAMCFG_INTPARTID registers. |

HAS_MSMON, bit [30]

Has resource Monitors. Indicates whether this MSC has MPAM resource monitors.

| HAS_MSMON | Meaning |
|------------------|---|
| 0b0 | Does not support MPAM resource monitoring by groups or MPAMF_MSMON_IDR . |
| 0b1 | Supports resource monitoring by matching a combination of PARTID and PMG. See MPAMF_MSMON_IDR . |

HAS_IMPL_IDR, bit [29]

Has [MPAMF_IMPL_IDR](#). Indicates whether this MSC has the IMPLEMENTATION SPECIFIC MPAM features register, [MPAMF_IMPL_IDR](#).

| HAS_IMPL_IDR | Meaning |
|---------------------|--|
| 0b0 | Does not have MPAMF_IMPL_IDR . |
| 0b1 | Has MPAMF_IMPL_IDR . |

EXT, bit [28]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Extended MPAMF_IDR.

| EXT | Meaning |
|------------|--|
| 0b0 | MPAMF_IDR has no defined bits in [63:32]. The register is effectively 32 bits. |
| 0b1 | MPAMF_IDR has bits defined in [63:32]. The register is 64-bits. |

Otherwise:

Reserved, RES0.

HAS_PRI_PART, bit [27]

Has Priority Partitioning. Indicates whether this MSC implements MPAM priority partitioning and [MPAMF_PRI_IDR](#).

| HAS_PRI_PART | Meaning |
|---------------------|--|
| 0b0 | Does not support priority partitioning or have MPAMF_PRI_IDR . |
| 0b1 | Has MPAMF_PRI_IDR . |

HAS_MBW_PART, bit [26]

Has Memory Bandwidth Partitioning. Indicates whether this MSC implements MPAM memory bandwidth partitioning and [MPAMF_MBW_IDR](#).

| HAS_MBW_PART | Meaning |
|--------------|--|
| 0b0 | Does not support memory bandwidth partitioning or have MPAMF_MBW_IDR register. |
| 0b1 | Has MPAMF_MBW_IDR register. |

HAS_CPOR_PART, bit [25]

Has Cache Portion Partitioning. Indicates whether this MSC implements MPAM cache portion partitioning and [MPAMF_CPOR_IDR](#).

| HAS_CPOR_PART | Meaning |
|---------------|--|
| 0b0 | Does not support cache portion partitioning or have MPAMF_CPOR_IDR or MPAMCFG_CPBm<n> registers. |
| 0b1 | Has MPAMF_CPOR_IDR and MPAMCFG_CPBm<n> registers. |

HAS_CCAP_PART, bit [24]

Has Cache Capacity Partitioning. Indicates whether this MSC implements MPAM cache capacity partitioning and the [MPAMF_CCAP_IDR](#) and [MPAMCFG_CMAX](#) registers.

| HAS_CCAP_PART | Meaning |
|---------------|---|
| 0b0 | Does not support cache capacity partitioning or have MPAMF_CCAP_IDR and MPAMCFG_CMAX registers. |
| 0b1 | Has MPAMF_CCAP_IDR and MPAMCFG_CMAX registers. |

PMG_MAX, bits [23:16]

Maximum supported value of PMG.

The value of this field is permitted to vary between the instances of [MPAM_IDR](#), each reporting the maximum supported PMG value in the PARTID space associated with that instance.

In [MPAMF_IDR_s](#) this field is permitted to report the maximum PMG value for the Non-secure PARTID space or for the Secure PARTID space. The maximum PMG value for the Secure PARTID space can be read from [MPAMF_SIDR.PMG_MAX](#).

PARTID_MAX, bits [15:0]

Maximum supported value of PARTID.

The value of this field is permitted to vary between the instances of [MPAM_IDR](#), each reporting the maximum supported PARTID value in the PARTID space associated with that instance.

In [MPAMF_IDR_s](#) this field is permitted to report the maximum PARTID value for the Non-secure PARTID space or for the Secure PARTID space. The maximum PARTID value for the Secure PARTID space can be read from [MPAMF_SIDR.PARTID_MAX](#).

Accessing MPAMF_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

[MPAMF_IDR](#) is read-only.

[MPAMF_IDR](#) must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

[MPAMF_IDR](#) is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- [MPAMF_IDR_s](#) is permitted to have either the same or different contents to [MPAMF_IDR_ns](#), [MPAMF_IDR_rt](#), or [MPAMF_IDR_rl](#).

- MPAMF_IDR_ns is permitted to have either the same or different contents to MPAMF_IDR_rt or MPAMF_IDR_rl.
- MPAMF_IDR_rt is permitted to have either the same or different contents to MPAMF_IDR_rl.

There must be separate registers in the Secure (MPAMF_IDR_s), Non-secure (MPAMF_IDR_ns), Root (MPAMF_IDR_rt), and Realm (MPAMF_IDR_rl) MPAM feature pages.

When [MPAMF_IDR.HAS_RIS](#) is 1, MPAMF_IDR shows the configuration of MSC MPAM for the resource instance selected by [MPAMCFG_PART_SEL.RIS](#). Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

MPAMF_IDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|-------------|
| MPAM | MPAMF_BASE_s | 0x0000 | MPAMF_IDR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------|
| MPAM | MPAMF_BASE_ns | 0x0000 | MPAMF_IDR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------|
| MPAM | MPAMF_BASE_rt | 0x0000 | MPAMF_IDR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------|
| MPAM | MPAMF_BASE_rl | 0x0000 | MPAMF_IDR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_IIDR, MPAM Implementation Identification Register

The MPAMF_IIDR characteristics are:

Purpose

Uniquely identifies the MSC implementation by the combination of implementer, product ID, variant, and revision.

Configuration

The power domain of MPAMF_IIDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_IIDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_IIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|----|---------|----|----|----|----------|----|----|----|-------------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ProductID | | | | | | | | | | | | Variant | | | | Revision | | | | Implementer | | | | | | | | | | | |

ProductID, bits [31:20]

The MSC implementer as identified in the MPAMF_IIDR.Implementer field must assure each product has a unique ProductID from any other with the same Implementer value.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Variant, bits [19:16]

This field distinguishes product variants or major revisions of the product.

Note

Implementations of ProductID with differing software interfaces are expected to have different values in the MPAMF_IIDR.Variant field.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Revision, bits [15:12]

This field distinguishes minor revisions of the product.

Note

This field is intended to differentiate product revisions that are minor changes and are largely software compatible with previous revisions.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Implementer, bits [11:0]

Contains the JEP106 code of the company that implemented the MPAM MSC.

[11:8] must contain the JEP106 continuation code of the implementer.

[7] must always be 0.

[6:0] must contain the JEP106 identity code of the implementer.

For an Arm implementation, bits[11:0] are 0x43B.

Accessing MPAMF_IIDR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_IIDR is read-only.

MPAMF_IIDR must be readable from the Secure, Non-secure, Root, and Realm MPAM feature pages.

MPAMF_IIDR must have the same contents in the Secure, Non-secure, Root, and Realm MPAM feature pages.

MPAMF_IIDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|--------------|
| MPAM | MPAMF_BASE_s | 0x0018 | MPAMF_IIDR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_ns | 0x0018 | MPAMF_IIDR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_rt | 0x0018 | MPAMF_IIDR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_rl | 0x0018 | MPAMF_IIDR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

MPAMF_IMPL_IDR, MPAM Implementation-Specific Partitioning Feature Identification Register

The MPAMF_IMPL_IDR characteristics are:

Purpose

Indicates the implementation-defined partitioning and monitoring features and parameters of the MSC.

MPAMF_IMPL_IDR_s indicates IMPLEMENTATION DEFINED partitioning and monitoring features accessed from the Secure MPAM feature page. MPAMF_IMPL_IDR_ns indicates those accessed from the Non-secure MPAM feature page. MPAMF_IMPL_IDR_rt indicates IMPLEMENTATION DEFINED partitioning and monitoring features accessed from the Root MPAM feature page. MPAMF_IMPL_IDR_rl indicates those accessed from the Realm MPAM feature page.

If [MPAMF_IDR.HAS_RIS](#) is 1, this register gives the implementation-specific features and parameters of the resource instance selected by [MPAMCFG_PART_SEL.RIS](#) for any features that are specific to the resource.

Configuration

The power domain of MPAMF_IMPL_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_IMPL_IDR == 1. Otherwise, direct accesses to MPAMF_IMPL_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_IMPL_IDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | IMPLFEAT | | | | | | | | | | | | | | | |

IMPLFEAT, bits [31:0]

All 32 bits of this register are available to be used as the implementer sees fit to indicate the presence of IMPLEMENTATION DEFINED MPAM features in this MSC and to give additional implementation-specific read-only information about the parameters of implementation-specific MPAM features to software.

If RIS is implemented, this register indicates the implementation-specific features and parameters of the resource instance selected by [MPAMCFG_PART_SEL.RIS](#).

Accessing MPAMF_IMPL_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_IMPL_IDR is read-only.

MPAMF_IMPL_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_IMPL_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_IMPL_IDR_s is permitted to have either the same or different contents to MPAMF_IMPL_IDR_ns, MPAMF_IMPL_IDR_rt, or MPAMF_IMPL_IDR_rl.

- MPAMF_IMPL_IDR_ns is permitted to have either the same or different contents to MPAMF_IMPL_IDR_rt or MPAMF_IMPL_IDR_rl.
- MPAMF_IMPL_IDR_rt is permitted to have either the same or different contents to MPAMF_IMPL_IDR_rl.

There must be separate registers in the Secure (MPAMF_IMPL_IDR_s), Non-secure (MPAMF_IMPL_IDR_ns), Root (MPAMF_IMPL_IDR_rt), and Realm (MPAMF_IMPL_IDR_rl) MPAM feature pages.

When [MPAMF_IDR.HAS_RIS](#) is 1, MPAMF_IMPL_IDR shows the configuration of implementation-specific features for the resource instance selected by [MPAMCFG_PART_SEL.RIS](#). Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

MPAMF_IMPL_IDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|------------------|
| MPAM | MPAMF_BASE_s | 0x0028 | MPAMF_IMPL_IDR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_ns | 0x0028 | MPAMF_IMPL_IDR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_rt | 0x0028 | MPAMF_IMPL_IDR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_rl | 0x0028 | MPAMF_IMPL_IDR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_MBW_IDR, MPAM Memory Bandwidth Partitioning Identification Register

The MPAMF_MBW_IDR characteristics are:

Purpose

Indicates which MPAM bandwidth partitioning features are present on this MSC.

MPAMF_MBW_IDR_s indicates bandwidth partitioning features accessed from the Secure MPAM feature page.
 MPAMF_MBW_IDR_ns indicates bandwidth partitioning features accessed from the Non-secure MPAM feature page.
 MPAMF_MBW_IDR_rt indicates bandwidth partitioning features accessed from the Root MPAM feature page.
 MPAMF_MBW_IDR_rl indicates bandwidth partitioning features accessed from the Realm MPAM feature page.

When [MPAMF_IDR.HAS_RIS](#) is 1, some fields in this register give information for the resource instance selected by [MPAMCFG_PART_SEL.RIS](#). The description of every field that is affected by [MPAMCFG_PART_SEL.RIS](#) has that information within the field description.

Configuration

The power domain of MPAMF_MBW_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_MBW_PART == 1. Otherwise, direct accesses to MPAMF_MBW_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_MBW_IDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:29]

Reserved, RES0.

BWPBM_WD, bits [28:16]

Bandwidth portion bitmap width.

The number of bandwidth portion bits in the [MPAMCFG_MBW_PBM<n>](#) register array.

If MPAMF_MBW_IDR.HAS_PBM is 1, this field must contain a value from 1 to 4096, inclusive. Values greater than 32 require a group of 32-bit registers to access the BWPBM, up to 128 if BWPBM_WD is the largest value.

If MPAMF_MBW_IDR.HAS_PBM is 0, this field must be ignored by software.

If RIS is implemented, this field indicates the width of the memory bandwidth portion bitmap partitioning control for the resource instance selected by [MPAMCFG_PART_SEL.RIS](#).

Bit [15]

Reserved, RES0.

WINDWR, bit [14]

Indicates the bandwidth accounting period register is writable.

| WINDWR | Meaning |
|---------------|---|
| 0b0 | The bandwidth accounting period is readable from MPAMCFG_MBW_WINWD which might be fixed or vary due to clock rate reconfiguration of the memory channel or memory controller. |
| 0b1 | The bandwidth accounting width is readable and writable per partition in MPAMCFG_MBW_WINWD . |

HAS_PROP, bit [13]

Indicates that this MSC implements proportional stride bandwidth partitioning and the [MPAMCFG_MBW_PROP](#) register can be accessed.

| HAS_PROP | Meaning |
|-----------------|--|
| 0b0 | There is no memory bandwidth proportional stride control and the MPAMCFG_MBW_PROP register is RES0. |
| 0b1 | The proportional stride memory bandwidth partitioning scheme is supported and the MPAMCFG_MBW_PROP register can be accessed. |

If RIS is implemented, this field indicates the presence of the memory bandwidth proportional stride partitioning control for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

HAS_PBM, bit [12]

Indicates that bandwidth portion partitioning is implemented and the [MPAMCFG_MBW_PBM<n>](#) register array can be accessed.

| HAS_PBM | Meaning |
|----------------|--|
| 0b0 | There is no memory bandwidth portion control and the MPAMCFG_MBW_PBM<n> is RES0. |
| 0b1 | The memory bandwidth portion allocation scheme exists and the MPAMCFG_MBW_PBM<n> register can be accessed. |

If RIS is implemented, this field indicates the presence of the memory bandwidth portion partitioning control for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

HAS_MAX, bit [11]

Indicates that this MSC implements maximum bandwidth partitioning and the [MPAMCFG_MBW_MAX](#) register can be accessed.

| HAS_MAX | Meaning |
|----------------|---|
| 0b0 | There is no maximum memory bandwidth control and the MPAMCFG_MBW_MAX register is RES0. |
| 0b1 | The maximum memory bandwidth allocation scheme is supported and the MPAMCFG_MBW_MAX register can be accessed. |

If RIS is implemented, this field indicates the presence of the maximum bandwidth partitioning control for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

HAS_MIN, bit [10]

Indicates that this MSC implements minimum bandwidth partitioning and the [MPAMCFG_MBW_MIN](#) register can be accessed.

| HAS_MIN | Meaning |
|---------|---|
| 0b0 | There is no minimum memory bandwidth control and the MPAMCFG_MBW_MIN register is RES0. |
| 0b1 | The minimum memory bandwidth allocation scheme is supported and the MPAMCFG_MBW_MIN register can be accessed. |

If RIS is implemented, this field indicates the presence of the minimum bandwidth partitioning control for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Bits [9:6]

Reserved, RES0.

BWA_WD, bits [5:0]

Number of implemented bits in the bandwidth allocation fields: MIN, MAX, and STRIDE. See [MPAMCFG_MBW_MIN](#), [MPAMCFG_MBW_MAX](#), and [MPAMCFG_MBW_PROP](#).

In any of these bandwidth allocation fields exist, this field must have a value from 1 to 16, inclusive. Otherwise, it is permitted to be 0.

If RIS is implemented, this field indicates the number of implemented bits in the bandwidth allocation control fields for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Accessing MPAMF_MBW_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_MBW_IDR is read-only.

MPAMF_MBW_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_MBW_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_MBW_IDR_s is permitted to have either the same or different contents to MPAMF_MBW_IDR_ns, MPAMF_MBW_IDR_rt, or MPAMF_MBW_IDR_rl.
- MPAMF_MBW_IDR_ns is permitted to have either the same or different contents to MPAMF_MBW_IDR_rt or MPAMF_MBW_IDR_rl.
- MPAMF_MBW_IDR_rt is permitted to have either the same or different contents to MPAMF_MBW_IDR_rl.

There must be separate registers in the Secure (MPAMF_MBW_IDR_s), Non-secure (MPAMF_MBW_IDR_ns), Root (MPAMF_MBW_IDR_rt), and Realm (MPAMF_MBW_IDR_rl) MPAM feature pages.

When [MPAMF_IDR](#).HAS_RIS is 1, MPAMF_MBW_IDR shows the configuration of memory bandwidth partitioning for the bandwidth resource instance selected by [MPAMCFG_PART_SEL](#).RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

MPAMF_MBW_IDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|-----------------|
| MPAM | MPAMF_BASE_s | 0x0040 | MPAMF_MBW_IDR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|------------------|
| MPAM | MPAMF_BASE_ns | 0x0040 | MPAMF_MBW_IDR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|-------|--------|----------|
|-----------|-------|--------|----------|

| | | | |
|------|---------------|--------|------------------|
| MPAM | MPAMF_BASE_rt | 0x0040 | MPAMF_MBW_IDR_rt |
|------|---------------|--------|------------------|

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|------------------|
| MPAM | MPAMF_BASE_rl | 0x0040 | MPAMF_MBW_IDR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_MBWUMON_IDR, MPAM Features Memory Bandwidth Usage Monitoring ID register

The MPAMF_MBWUMON_IDR characteristics are:

Purpose

Indicates the number of memory bandwidth usage monitor instances implemented. This register also indicates several properties of MBWU monitoring, including whether the implementation supports capture, scaling, or long counters.

MPAMF_MBWUMON_IDR_s indicates the number of Secure memory bandwidth usage monitor instances.

MPAMF_MBWUMON_IDR_ns indicates the number of Non-secure memory bandwidth usage monitor instances.

MPAMF_MBWUMON_IDR_rt indicates the number of Root memory bandwidth usage monitor instances.

MPAMF_MBWUMON_IDR_rl indicates the number of Realm memory bandwidth usage monitor instances.

If [MPAMF_IDR.HAS_RIS](#) is 1, fields that mention RIS must reflect the properties of the resource instance currently selected by [MPAMCFG_PART_SEL.RIS](#). Fields that do not mention RIS are constant across all resource instances.

Configuration

The power domain of MPAMF_MBWUMON_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MPAMF_MBWUMON_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_MBWUMON_IDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|--------------------------|---------------------|--------------------------|--------------------------------|--------------------------|--------------------------------|----------------------|-----------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| HAS_CAPTURE | HAS_LONG | LWD | HAS_RWBW | HAS_OFLOW_LNKG | HAS_OFSR | HAS_OFLOW_CAPT | RES0 | SCALE | | | | | | | | | | | | | | | | | | | | | | | |

HAS_CAPTURE, bit [31]

The implementation supports copying an [MSMON_MBWU](#) to the corresponding [MSMON_MBWU_CAPTURE](#) on a capture event.

| HAS_CAPTURE | Meaning |
|-------------|--|
| 0b0 | MSMON_MBWU_CAPTURE is not implemented and there is no support for capture events in the MBWU monitor. |
| 0b1 | The MSMON_MBWU_CAPTURE register is implemented and the MBWU monitor supports the capture event behavior. |

If RIS is implemented, this field indicates that MBWU monitor capture is implemented for the resource instance selected by [MPAMCFG_PART_SEL.RIS](#).

If MPAMF_MBWUMON_IDR.HAS_LONG is 1, this also indicates that [MSMON_MBWU_L_CAPTURE](#) is implemented.

HAS_LONG, bit [30]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Indicates whether [MSMON_MBWU_L](#) is implemented.

If HAS_CAPTURE is 1, indicates whether [MSMON_MBWU_L_CAPTURE](#) is implemented.

| HAS_LONG | Meaning |
|----------|--|
| 0b0 | Does not implement MSMON_MBWU_L or MSMON_MBWU_L_CAPTURE . |
| 0b1 | Implements MSMON_MBWU_L . If HAS_CAPTURE == 1, MSMON_MBWU_L_CAPTURE is also implemented. |

If RIS is implemented, this field indicates that the long MBWU monitor is implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

If MPAMF_MBWUMON_IDR.HAS_CAPTURE is 1, this also indicates that [MSMON_MBWU_L_CAPTURE](#) is implemented.

Otherwise:

Reserved, RES0.

LWD, bit [29]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Long register VALUE width.

If [MPAMF_MBWUMON_IDR](#).HAS_LONG is 0, [MPAMF_MBWUMON_IDR](#).LWD must also be 0.

| LWD | Meaning |
|-----|--|
| 0b0 | If MPAMF_MBWUMON_IDR .HAS_LONG is 1, MSMON_MBWU_L has 44-bit VALUE field in bits [43:0]. Bits [62:44] are RES0. If HAS_LONG is 1 and MPAMF_MBWUMON_IDR .HAS_CAPTURE is 1, MSMON_MBWU_L_CAPTURE also has 44-bit VALUE field in bits [43:0]. |
| 0b1 | MSMON_MBWU_L has 63-bit VALUE field in bits [62:0]. If MPAMF_MBWUMON_IDR .HAS_CAPTURE == 1, MSMON_MBWU_L_CAPTURE also has 63-bit VALUE field in bits [62:0]. |

If RIS is implemented, this field indicates the length of the [MSMON_MBWU_L](#).VALUE field implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

HAS_RWBW, bit [28]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Read/write bandwidth selection is implemented in [MSMON_CFG_MBWU_FLT](#).

| HAS_RWBW | Meaning |
|----------|--|
| 0b0 | Read/write bandwidth selection is not implemented. |
| 0b1 | Read/write bandwidth selection is implemented. |

If RIS is implemented, this field indicates whether read/write bandwidth collection selection is available in [MSMON_CFG_MBWU_FLT](#) for resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

HAS_OFLOW_LNKG, bit [27]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Supports [MSMON_CFG_MBWU_CTL](#).OFLOW_LNKG field to control how overflow on an instance affects other monitor instances in this MSC.

| HAS_OFLOW_LNKG | Meaning |
|----------------|--|
| 0b0 | Does not support MBWU overflow linkage. |
| 0b1 | Supports MBWU overflow linkage and the MSMON_CFG_MBWU_CTL .OFLOW_LNKG field. |

If RIS is implemented, this field indicates that [MSMON_CFG_MBWU_CTL](#).OFLOW_LNKG is implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

HAS_OFSR, bit [26]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

The MBWU monitor overflow status bitmap register, [MSMON_MBWU_OFSR](#), is implemented.

| HAS_OFSR | Meaning |
|----------|--|
| 0b0 | MSMON_MBWU_OFSR register is not implemented. |
| 0b1 | MSMON_MBWU_OFSR register is implemented. |

If RIS is implemented, this field indicates that MBWU monitor overflow status bitmap register is implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

HAS_OFLOW_CAPT, bit [25]

Supports [MSMON_CFG_MBWU_CTL](#).OFLOW_CAPT field to transfer the MBWU monitor instance to its capture register on an overflow or overflow linkage event.

| HAS_OFLOW_CAPT | Meaning |
|----------------|---|
| 0b0 | Does not support MBWU capture on overflow. |
| 0b1 | Supports MBWU capture on overflow and the MSMON_CFG_MBWU_CTL .OFLOW_CAPT field. |

If RIS is implemented, this field indicates that [MSMON_CFG_MBWU_CTL](#).OFLOW_CAPT is implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Bits [24:21]

Reserved, RES0.

SCALE, bits [20:16]

Scaling of [MSMON_MBWU.VALUE](#) in bits. If scaling is enabled by [MSMON_CFG_MBWU_CTL.SCLEN](#), the byte count in the VALUE field has been shifted by SCALE bits to the right.

| SCALE | Meaning |
|---------|---|
| 0b00000 | Scaling is not implemented. |
| 0bxxxxx | Other values are right shift count when scaling is enabled. |

If RIS is implemented, this field indicates the scale value for [MSMON_MBWU.VALUE](#) field for the resource instance selected by [MPAMCFG_PART_SEL.RIS](#).

NUM_MON, bits [15:0]

The number of memory bandwidth usage monitor instances implemented. The largest monitor instance selector, [MSMON_CFG_MON_SEL.MON_SEL](#), is NUM_MON minus 1.

If RIS is implemented, this field indicates the number of MBWU monitor instances for [MSMON_MBWU.VALUE](#) field for the resource instance selected by [MPAMCFG_PART_SEL.RIS](#).

Accessing MPAMF_MBWUMON_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_MBWUMON_IDR is read-only.

MPAMF_MBWUMON_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_MBWUMON_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_MBWUMON_IDR_s is permitted to have either the same or different contents to MPAMF_MBWUMON_IDR_ns, MPAMF_MBWUMON_IDR_rt, or MPAMF_MBWUMON_IDR_rl.
- MPAMF_MBWUMON_IDR_ns is permitted to have either the same or different contents to MPAMF_MBWUMON_IDR_rt or MPAMF_MBWUMON_IDR_rl.
- MPAMF_MBWUMON_IDR_rt is permitted to have either the same or different contents to MPAMF_MBWUMON_IDR_rl.

There must be separate registers in the Secure (MPAMF_MBWUMON_IDR_s), Non-secure (MPAMF_MBWUMON_IDR_ns), Root (MPAMF_MBWUMON_IDR_rt), and Realm (MPAMF_MBWUMON_IDR_rl) MPAM feature pages.

When [MPAMF_IDR.HAS_RIS](#) is 1, MPAMF_MBWUMON_IDR shows the configuration of memory bandwidth monitoring for the bandwidth resource instance selected by [MPAMCFG_PART_SEL.RIS](#). Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

Access to MPAMF_MBWUMON_IDR is not affected by [MSMON_CFG_MON_SEL.RIS](#).

MPAMF_MBWUMON_IDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|---------------------|
| MPAM | MPAMF_BASE_s | 0x0090 | MPAMF_MBWUMON_IDR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_ns | 0x0090 | MPAMF_MBWUMON_IDR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_rt | 0x0090 | MPAMF_MBWUMON_IDR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_rl | 0x0090 | MPAMF_MBWUMON_IDR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_MSMON_IDR, MPAM Resource Monitoring Identification Register

The MPAMF_MSMON_IDR characteristics are:

Purpose

Indicates which MPAM monitoring features are present on this MSC.

MPAMF_MSMON_IDR_s indicates Secure monitoring features. MPAMF_MSMON_IDR_ns indicates Non-secure monitoring features. MPAMF_MSMON_IDR_rt indicates Root monitoring features. MPAMF_MSMON_IDR_rl indicates Realm monitoring features.

If [MPAMF_IDR.HAS_RIS](#) is 1, fields that mention RIS must reflect the properties of the resource instance currently selected by [MPAMCFG_PART_SEL.RIS](#). Fields that do not mention RIS are constant across all resource instances.

Configuration

The power domain of MPAMF_MSMON_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_MSMON == 1. Otherwise, direct accesses to MPAMF_MSMON_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_MSMON_IDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | |
|---------------------|-----------------|-------------|--------------|------|----|----|----|----|----|-------------|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 |
| HAS_LOCAL_CAPT_EVNT | NO_HW_OFLW_INTR | HAS_OFLW_MS | HAS_OFLOW_SR | RES0 | | | | | | MSMON_MBWUM | | | | |

HAS_LOCAL_CAPT_EVNT, bit [31]

Has local capture event generator. Indicates whether this MSC has the MPAM local capture event generator and the [MSMON_CAPT_EVNT](#) register.

| HAS_LOCAL_CAPT_EVNT | Meaning |
|---------------------|---|
| 0b0 | Does not support MPAM local capture event generator or MSMON_CAPT_EVNT . |
| 0b1 | Supports the MPAM local capture event generator and the MSMON_CAPT_EVNT register. |

NO_HW_OFLW_INTR, bit [30]

When FEAT_MPAMv1p1 is implemented:

Does not have hardwired MPAM monitor overflow interrupt.

| NO_HW_OFLW_INTR | Meaning |
|-----------------|--|
| 0b0 | Supports generating a hardwired interrupt to signal MPAM monitor overflow. |
| 0b1 | No support for a hardwired interrupt to signal MPAM monitor overflow. |

If this field is 0, the MSC supports generating a hardwired interrupt for monitor overflow events.

If this field is 0 and the HAS_OFLW_MSI field in this register is 1, the MSC supports generating both hardwired interrupts and MSI writes to signal interrupts.

Otherwise:

Reserved, RES0.

HAS_OFLW_MSI, bit [29]

When FEAT_MPAMv1p1 is implemented:

Has support for MSI writes to signal MPAM monitor overflow interrupts. These registers are implemented: [MSMON_OFLOW_MSI_ADDR_L](#), [MSMON_OFLOW_MSI_ADDR_H](#), [MSMON_OFLOW_MSI_ATTR](#), [MSMON_OFLOW_MSI_DATA](#) and [MSMON_OFLOW_MSI_MPAM](#).

| HAS_OFLW_MSI | Meaning |
|--------------|---|
| 0b0 | MSMON_OFLOW_MSI_ADDR_L , MSMON_OFLOW_MSI_ADDR_H , MSMON_OFLOW_MSI_ATTR , MSMON_OFLOW_MSI_DATA and MSMON_OFLOW_MSI_MPAM registers are not implemented. |
| 0b1 | MSMON_OFLOW_MSI_ADDR_L , MSMON_OFLOW_MSI_ADDR_H , MSMON_OFLOW_MSI_ATTR , MSMON_OFLOW_MSI_DATA and MSMON_OFLOW_MSI_ATTR are implemented and can be used to generate writes to signal MPAM monitor overflow interrupts. |

If [MPAMF_MSMON_IDR.NO_HW_OFLW_INTR](#) is 1 and this bit is 0, this MSC does not support monitor overflow interrupts.

Otherwise:

Reserved, RES0.

HAS_OFLOW_SR, bit [28]

When FEAT_MPAMv1p1 is implemented:

Has MPAM monitor overflow status register [MSMON_OFLOW_SR](#).

| HAS_OFLOW_SR | Meaning |
|--------------|--|
| 0b0 | Does not have MSMON_OFLOW_SR . |
| 0b1 | Supports MSMON_OFLOW_SR . |

Otherwise:

Reserved, RES0.

Bits [27:18]

Reserved, RES0.

MSMON_MBWU, bit [17]

Memory bandwidth usage monitoring. Indicates whether MPAM monitoring for Memory Bandwidth Usage by PARTID and PMG is implemented and whether the following bandwidth usage registers are accessible:

- [MPAMF_MBWUMON_IDR](#), [MSMON_CFG_MBWU_CTL](#), [MSMON_CFG_MBWU_FLT](#), [MSMON_MBWU](#).
- The optional [MSMON_MBWU_CAPTURE](#).
- If MPAM v0.1 or MPAM v1.1 is implemented, the optional [MSMON_MBWU_L](#) and the optional [MSMON_MBWU_L_CAPTURE](#).

| MSMON_MBWU | Meaning |
|------------|---|
| 0b0 | Does not have monitoring for memory bandwidth usage and does not use the bandwidth usage registers. |
| 0b1 | Has monitoring of memory bandwidth usage and uses the bandwidth usage registers. |

If RIS is implemented, this field indicates that memory bandwidth usage monitoring is implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS as described in [MPAMF_MBWUMON_IDR](#).

MSMON_CSU, bit [16]

Cache storage usage monitoring. Indicates whether MPAM monitoring of cache storage usage by PARTID and PMG is implemented and the following registers are accessible:

- [MPAMF_CSUMON_IDR](#), [MSMON_CFG_CSU_CTL](#), [MSMON_CFG_CSU_FLT](#), [MSMON_CSU](#).
- The optional [MSMON_CSU_CAPTURE](#).

| MSMON_CSU | Meaning |
|-----------|--|
| 0b0 | Does not have monitoring for cache storage usage or the MPAMF_CSUMON_IDR , MSMON_CFG_CSU_CTL , MSMON_CFG_CSU_FLT , MSMON_CSU or MSMON_CSU_CAPTURE registers. |
| 0b1 | Has monitoring of cache storage usage and the MPAMF_CSUMON_IDR , MSMON_CFG_CSU_CTL , MSMON_CFG_CSU_FLT , MSMON_CSU and optional MSMON_CSU_CAPTURE registers. |

If RIS is implemented, this field indicates that cache storage usage monitoring is implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS as described in [MPAMF_CSUMON_IDR](#).

Bits [15:0]

Reserved, RES0.

Accessing MPAMF_MSMON_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_MSMON_IDR is read-only.

MPAMF_MSMON_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_MSMON_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_MSMON_IDR_s is permitted to have either the same or different contents to MPAMF_MSMON_IDR_ns, MPAMF_MSMON_IDR_rt, or MPAMF_MSMON_IDR_rl.
- MPAMF_MSMON_IDR_ns is permitted to have either the same or different contents to MPAMF_MSMON_IDR_rt or MPAMF_MSMON_IDR_rl.
- MPAMF_MSMON_IDR_rt is permitted to have either the same or different contents to MPAMF_MSMON_IDR_rl.

There must be separate registers in the Secure (MPAMF_MSMON_IDR_s), Non-secure (MPAMF_MSMON_IDR_ns), Root (MPAMF_MSMON_IDR_rt), and Realm (MPAMF_MSMON_IDR_rl) MPAM feature pages.

When [MPAMF_IDR](#).HAS_RIS is 1, MPAMF_MSMON_IDR shows the configuration of memory system monitoring for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

Access to MPAMF_MSMON_IDR is not affected by [MSMON_CFG_MON_SEL](#).RIS.

MPAMF_MSMON_IDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|-------------------|
| MPAM | MPAMF_BASE_s | 0x0080 | MPAMF_MSMON_IDR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_ns | 0x0080 | MPAMF_MSMON_IDR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_rt | 0x0080 | MPAMF_MSMON_IDR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_rl | 0x0080 | MPAMF_MSMON_IDR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MPAMF_PARTID_NRW_IDR, MPAM PARTID Narrowing ID register

The MPAMF_PARTID_NRW_IDR characteristics are:

Purpose

Indicates the largest internal PARTID for this MSC.

MPAMF_PARTID_NRW_IDR_s indicates the largest Secure internal PARTID. MPAMF_PARTID_NRW_IDR_ns indicates the largest Non-secure internal PARTID.

When FEAT_RME is implemented: MPAMF_PARTID_NRW_rt indicates the largest Root internal PARTID. MPAMF_PARTID_NRW_rl indicates the largest Realm internal PARTID.

PARTID narrowing is global to the MSC and does not vary by resource instance.

Configuration

The power domain of MPAMF_PARTID_NRW_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_PARTID_NRW == 1. Otherwise, direct accesses to MPAMF_PARTID_NRW_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_PARTID_NRW_IDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | INTPARTID_MAX | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

INTPARTID_MAX, bits [15:0]

The largest intPARTID supported in this MSC.

Accessing MPAMF_PARTID_NRW_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_PARTID_NRW_IDR is read-only.

MPAMF_PARTID_NRW_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_PARTID_NRW_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_PARTID_NRW_IDR_s is permitted to have either the same or different contents to MPAMF_PARTID_NRW_IDR_ns, MPAMF_PARTID_NRW_IDR_rt, or MPAMF_PARTID_NRW_IDR_rl.

- MPAMF_PARTID_NRW_IDR_ns is permitted to have either the same or different contents to MPAMF_PARTID_NRW_IDR_rt or MPAMF_PARTID_NRW_IDR_rl.
- MPAMF_PARTID_NRW_IDR_rt is permitted to have either the same or different contents to MPAMF_PARTID_NRW_IDR_rl.

There must be separate registers in the Secure (MPAMF_PARTID_NRW_IDR_s), Non-secure (MPAMF_PARTID_NRW_IDR_ns), Root (MPAMF_PARTID_NRW_IDR_rt), and Realm (MPAMF_PARTID_NRW_IDR_rl) MPAM feature pages.

MPAMF_PARTID_NRW_IDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|------------------------|
| MPAM | MPAMF_BASE_s | 0x0050 | MPAMF_PARTID_NRW_IDR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_ns | 0x0050 | MPAMF_PARTID_NRW_IDR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_rt | 0x0050 | MPAMF_PARTID_NRW_IDR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_rl | 0x0050 | MPAMF_PARTID_NRW_IDR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

MPAMF_PRI_IDR, MPAM Priority Partitioning Identification Register

The MPAMF_PRI_IDR characteristics are:

Purpose

Indicates which MPAM priority partitioning features are present on this MSC.

MPAMF_PRI_IDR_s indicates priority partitioning features accessed from the Secure MPAM feature page.

MPAMF_PRI_IDR_ns indicates priority partitioning features accessed from the Non-secure MPAM feature page.

MPAMF_PRI_IDR_rt indicates priority partitioning features accessed from the Root MPAM feature page.

MPAMF_PRI_IDR_rl indicates priority partitioning features accessed from the Realm MPAM feature page.

When MPAMF_IDR.HAS_RIS is 1, some fields in this register give information for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS. The description of every field that is affected by [MPAMCFG_PART_SEL](#).RIS has that information within the field description.

Configuration

The power domain of MPAMF_PRI_IDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and MPAMF_IDR.HAS_PRI_PART == 1. Otherwise, direct accesses to MPAMF_PRI_IDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_PRI_IDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----------|------|----------------|-----------|------|-----------|------|-----------------|------------|------|-----------------|------------|------|-----------------|------------|------|-----------------|------------|------|-----------------|------------|------|-----------------|------------|------|-----------------|------------|------|-----------------|------------|------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | DSPRI_WD | RES0 | DSPRI_0_IS_LOW | HAS_DSPRI | RES0 | INTPRI_WD | RES0 | INTPRI_0_IS_LOW | HAS_INTPRI | RES0 | INTPRI_0_IS_LOW | HAS_INTPRI | RES0 | INTPRI_0_IS_LOW | HAS_INTPRI | RES0 | INTPRI_0_IS_LOW | HAS_INTPRI | RES0 | INTPRI_0_IS_LOW | HAS_INTPRI | RES0 | INTPRI_0_IS_LOW | HAS_INTPRI | RES0 | INTPRI_0_IS_LOW | HAS_INTPRI | RES0 | INTPRI_0_IS_LOW | HAS_INTPRI | RES0 |

Bits [31:26]

Reserved, RES0.

DSPRI_WD, bits [25:20]

Number of implemented bits in the downstream priority field (DSPRI) of [MPAMCFG_PRI](#).

If HAS_DSPRI == 1, this field must contain a value from 1 to 16, inclusive.

If HAS_DSPRI == 0, this field must be 0.

If RIS is implemented, this field indicates the number of downstream priority bits for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Bits [19:18]

Reserved, RES0.

DSPRI_0_IS_LOW, bit [17]

Indicates whether 0 in [MPAMCFG_PRI](#).DSPRI is the lowest or the highest downstream priority.

| DSPRI_0_IS_LOW | Meaning |
|-----------------------|---|
| 0b0 | In the MPAMCFG_PRI .DSPRI field, a value of 0 means the highest priority. |
| 0b1 | In the MPAMCFG_PRI .DSPRI field, a value of 0 means the lowest priority. |

If RIS is implemented, this field indicates that 0 is the lowest downstream priority for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

HAS_DSPRI, bit [16]

Indicates that the [MPAMCFG_PRI](#) register implements the DSPRI field.

| HAS_DSPRI | Meaning |
|------------------|--|
| 0b0 | This MSC supports priority partitioning, but does not implement a downstream priority (DSPRI) field in the MPAMCFG_PRI register. |
| 0b1 | This MSC supports downstream priority partitioning and implements the downstream priority (DSPRI) field in the MPAMCFG_PRI register. |

If RIS is implemented, this field indicates that downstream priority is implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Bits [15:10]

Reserved, RES0.

INTPRI_WD, bits [9:4]

Number of implemented bits in the internal priority field (INTPRI) in the [MPAMCFG_PRI](#) register.

If HAS_INTPRI == 1, this field must contain a value from 1 to 16, inclusive.

If HAS_INTPRI == 0, this field must be 0.

If RIS is implemented, this field indicates the number of internal priority bits for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Bits [3:2]

Reserved, RES0.

INTPRI_0_IS_LOW, bit [1]

Indicates whether 0 in [MPAMCFG_PRI](#).INTPRI is the lowest or the highest internal priority.

| INTPRI_0_IS_LOW | Meaning |
|------------------------|--|
| 0b0 | In the MPAMCFG_PRI .INTPRI field, a value of 0 means the highest priority. |
| 0b1 | In the MPAMCFG_PRI .INTPRI field, a value of 0 means the lowest priority. |

If RIS is implemented, this field indicates that 0 is the lowest internal priority for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

HAS_INTPRI, bit [0]

Indicates that this MSC implements the INTPRI field in the [MPAMCFG_PRI](#) register.

| HAS_INTPRI | Meaning |
|------------|---|
| 0b0 | This MSC supports priority partitioning, but does not implement the internal priority (INTPRI) field in the MPAMCFG_PRI register. |
| 0b1 | This MSC supports internal priority partitioning and implements the internal priority (INTPRI) field in the MPAMCFG_PRI register. |

If RIS is implemented, this field indicates that internal priority is implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Accessing MPAMF_PRI_IDR

This register is within the MPAM feature page memory frames. In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps.

MPAMF_PRI_IDR is read-only.

MPAMF_PRI_IDR must be readable from the Non-secure, Secure, Root, and Realm MPAM feature pages.

MPAMF_PRI_IDR is permitted to have the same contents when read from the Secure, Non-secure, Root, and Realm MPAM feature pages unless the register contents are different for the different versions:

- MPAMF_PRI_IDR_s is permitted to have either the same or different contents to MPAMF_PRI_IDR_ns, MPAMF_PRI_IDR_rt, or MPAMF_PRI_IDR_rl.
- MPAMF_PRI_IDR_ns is permitted to have either the same or different contents to MPAMF_PRI_IDR_rt or MPAMF_PRI_IDR_rl.
- MPAMF_PRI_IDR_rt is permitted to have either the same or different contents to MPAMF_PRI_IDR_rl.

There must be separate registers in the Secure (MPAMF_PRI_IDR_s), Non-secure (MPAMF_PRI_IDR_ns), Root (MPAMF_PRI_IDR_rt), and Realm (MPAMF_PRI_IDR_rl) MPAM feature pages.

When [MPAMF_IDR](#).HAS_RIS is 1, MPAMF_PRI_IDR shows the configuration of priority partitioning for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS. Fields that mention RIS in their field descriptions have values that track the implemented properties of the resource instance. Fields that do not mention RIS are constant across all resource instances.

MPAMF_PRI_IDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|-----------------|
| MPAM | MPAMF_BASE_s | 0x0048 | MPAMF_PRI_IDR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|------------------|
| MPAM | MPAMF_BASE_ns | 0x0048 | MPAMF_PRI_IDR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|------------------|
| MPAM | MPAMF_BASE_rt | 0x0048 | MPAMF_PRI_IDR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|------------------|
| MPAM | MPAMF_BASE_rl | 0x0048 | MPAMF_PRI_IDR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

MPAMF_SIDR, MPAM Features Secure Identification Register

The MPAMF_SIDR characteristics are:

Purpose

The MPAMF_SIDR is a 32-bit read-only register that indicates the maximum Secure PARTID and Secure PMG on this MSC.

Configuration

The power domain of MPAMF_SIDR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented. Otherwise, direct accesses to MPAMF_SIDR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MPAMF_SIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-----------|----|----|----|----|----|----|----|--------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | S_PMG_MAX | | | | | | | | S_PARTID_MAX | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

S_PMG_MAX, bits [23:16]

Maximum value of Secure PMG supported by this component.

S_PARTID_MAX, bits [15:0]

Maximum value of Secure PARTID supported by this component.

Accessing MPAMF_SIDR

This register is only within the Secure MPAM feature page memory frame.

MPAMF_SIDR is read-only.

MPAMF_SIDR must only be readable from the Secure MPAM feature page. If the system or the MSC does not support the Secure address map, this register must not be accessible.

MPAMF_SIDR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|--------------|
| MPAM | MPAMF_BASE_s | 0x0008 | MPAMF_SIDR_s |

Accesses on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_CAPT_EVNT, MPAM Capture Event Generation Register

The MSMON_CAPT_EVNT characteristics are:

Purpose

Generates a local capture event when written with bit[0] as 1.

MSMON_CAPT_EVNT_s generates local capture events for Secure monitor instances only or for Secure and Non-secure monitor instances. MSMON_CAPT_EVNT_ns generates local capture events for Non-secure monitor instances only. MSMON_CAPT_EVNT_rt generates local capture events for Root monitor instances only or for Root, Secure, Realm, and Non-secure monitor instances. MSMON_CAPT_EVNT_rl generates local capture events for Realm monitor instances or for for Realm monitor instances or Realm and Non-secure monitor instances.

Configuration

The power domain of MSMON_CAPT_EVNT is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.HAS_LOCAL_CAPT_EVNT == 1. Otherwise, direct accesses to MSMON_CAPT_EVNT are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CAPT_EVNT is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|-----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | ALL | | NOW | | | | | | | | | | | | | |

Bits [31:2]

Reserved, RES0.

ALL, bit [1]

In the Secure instance of this register:

- If ALL is written as 1 and NOW is also written as 1, signal a capture event to Secure and Non-secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.
- If ALL is written as 0 and NOW is written as 1, signal a capture event to Secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.

In the Non-secure instance of this register, this bit is RAZ/WI.

In the Root instance of this register:

- If ALL is written as 1 and NOW is also written as 1, signal a capture event to Root, Realm, Secure, and Non-secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.
- If ALL is written as 0 and NOW is written as 1, signal a capture event to Root monitor instances within this MSC that are configured with CAPT_EVNT = 7.

In the Realm instance of this register:

- If ALL is written as 1 and NOW is also written as 1, signal a capture event to Realm and Non-secure monitor instances in this MSC that are configured with CAPT_EVNT = 7.

- If ALL is written as 0 and NOW is written as 1, signal a capture event to Realm monitor instances within this MSC that are configured with CAPT_EVNT = 7.

This bit always reads as zero.

| ALL | Meaning |
|-----|---|
| 0b0 | Send capture event only to monitor instances in the same MPAM feature page as this register. |
| 0b1 | Send capture event to monitor instances in certain MPAM feature pages as described in the ALL field of this register. |

NOW, bit [0]

When written as 1, this bit causes an event to those monitor instances described in the ALL field that have CAPT_EVNT set to the value of 7.

When this bit is written as 0, no event is signaled.

This bit always reads as zero.

Accessing MSMON_CAPT_EVNT

This register is within the MPAM feature page memory frames. In a system that supports Secure and Non-secure memory maps, there must be both Secure and Non-secure MPAM feature pages.

MSMON_CAPT_EVNT_s must be accessible from the Secure MPAM feature page. MSMON_CAPT_EVNT_ns must be accessible from the Non-secure MPAM feature page.

MSMON_CAPT_EVNT_s and MSMON_CAPT_EVNT_ns must be separate registers. The Secure instance (MSMON_CAPT_EVNT_s) can generate local capture events for Secure monitor instances only or for Secure and Non-secure monitor instances, and the Non-secure instance (MSMON_CAPT_EVNT_ns) can generate local capture events for Non-secure monitor instances only.

MSMON_CAPT_EVNT can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|-------------------|
| MPAM | MPAMF_BASE_s | 0x0808 | MSMON_CAPT_EVNT_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_ns | 0x0808 | MSMON_CAPT_EVNT_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_rt | 0x0808 | MSMON_CAPT_EVNT_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_rl | 0x0808 | MSMON_CAPT_EVNT_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MSMON_CFG_CSU_CTL, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Control Register

The MSMON_CFG_CSU_CTL characteristics are:

Purpose

Controls the CSU monitor selected by [MSMON_CFG_MON_SEL](#).

MSMON_CFG_CSU_CTL_s controls the Secure cache storage usage monitor instance selected by the Secure instance of [MSMON_CFG_MON_SEL](#). MSMON_CFG_CSU_CTL_ns controls Non-secure cache storage usage monitor instance selected by the Non-secure instance of [MSMON_CFG_MON_SEL](#). MSMON_CFG_CSU_CTL_rt controls the monitor configuration for the Root PARTID selected by the Root instance of [MSMON_CFG_MON_SEL](#). MSMON_CFG_CSU_CTL_rl controls the monitor configuration for the Realm PARTID selected by the Realm instance of [MSMON_CFG_MON_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the monitor instance configuration accessed is for the resource instance currently selected by [MSMON_CFG_MON_SEL](#).RIS and the monitor instance of that resource instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

Configuration

The power domain of MSMON_CFG_CSU_CTL is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CFG_CSU_CTL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_CSU_CTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | |
|--------------------|---------------------------|----------------------------|------------------------------|----------------------------|---------------------------|----------------------------|-------------------------|----------------------|----------------------------|---------------------|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 |
| EN | CAPT_EVNT | CAPT_RESET | OFLOW_STATUS | OFLOW_INTR | OFLOW_FRZ | OFLOW_CAPT | SUBTYPE | RES0 | CEVNT_OFLW | MAT | | | |

EN, bit [31]

Enabled.

| EN | Meaning |
|-----|--|
| 0b0 | The monitor instance is disabled and must not collect any information. |
| 0b1 | The monitor instance is enabled to collect information according to the configuration of the instance. |

CAPT_EVNT, bits [30:28]

Capture event selector.

Select the event that triggers capture from the following:

| CAPT_EVNT | Meaning |
|------------------|---|
| 0b000 | No capture event is triggered. |
| 0b001 | External capture event 1 (optional, but recommended) |
| 0b010 | External capture event 2 (optional) |
| 0b011 | External capture event 3 (optional) |
| 0b100 | External capture event 4 (optional) |
| 0b101 | External capture event 5 (optional) |
| 0b110 | External capture event 6 (optional) |
| 0b111 | Capture occurs when a MSMON_CAPT_EVNT register in this MSC is written and causes a capture event for the Security state of this monitor. (optional) |

The values marked as optional indicate capture event sources that can be omitted in an implementation. Those values representing non-implemented event sources must not trigger a capture event.

If capture is not implemented for the CSU monitor type as indicated by [MPAMF_CSUMON_IDR](#).HAS_CAPTURE = 0, this field is RAZ/WI.

CAPT_RESET, bit [27]

Reset after capture.

Controls whether the value of [MSMON_CSU](#) is reset to zero immediately after being copied to [MSMON_CSU_CAPTURE](#).

| CAPT_RESET | Meaning |
|-------------------|----------------------------------|
| 0b0 | Monitor is not reset on capture. |
| 0b1 | Monitor is reset on capture. |

If capture is not implemented for the CSU monitor type as indicated by [MPAMF_CSUMON_IDR](#).HAS_CAPTURE = 0, this field is RAZ/WI.

Because the CSU monitor type produces a measurement rather than a count, it might not make sense to ever reset the value after a capture. If there is no reason to ever reset a CSU monitor, this field is RAZ/WI.

OFLOW_STATUS, bit [26]

Overflow status.

Indicates whether the value of [MSMON_CSU](#) has overflowed.

If [MPAMF_CSUMON_IDR](#).HAS_OFLOW_CAPT is 1 or [MPAMF_CSUMON_IDR](#).HAS_OFLOW_LNKG is 1, then a store to [MSMON_CSU](#) when this field is 1 resets this field to 0.

| OFLOW_STATUS | Meaning |
|---------------------|---|
| 0b0 | No overflow has occurred. |
| 0b1 | At least one overflow has occurred since this bit was last written to zero. |

If overflow is not possible for a CSU monitor in the implementation, this field is RAZ/WI.

OFLOW_INTR, bit [25]

Overflow Interrupt.

Controls whether an overflow interrupt is generated when the value of [MSMON_CSU](#) has overflowed.

| OFLOW_INTR | Meaning |
|-------------------|--|
| 0b0 | No interrupt is signaled on an overflow of MSMON_CSU . |
| 0b1 | On overflow, an implementation-specific interrupt is signaled. |

If OFLOW_INTR is not supported by the implementation, this field is RAZ/WI.

OFLOW_FRZ, bit [24]

Freeze Monitor on Overflow.

Controls whether the value of [MSMON_CSU](#) freezes on an overflow.

| OFLOW_FRZ | Meaning |
|-----------|--|
| 0b0 | Monitor count wraps on overflow. |
| 0b1 | Monitor count freezes on overflow. The frozen value might be 0 or another value if the monitor overflowed with an increment larger than 1. |

If overflow is not possible for a CSU monitor in the implementation, this field is RAZ/WI.

OFLOW_CAPT, bit [23]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_CSUMON_IDR.HAS_OFLOW_CAPT == 1:

Capture Monitor on Overflow.

Controls whether the value of [MSMON_CSU](#) is captured on an overflow or an overflow linkage event.

| OFLOW_CAPT | Meaning |
|------------|---|
| 0b0 | Monitor is not captured on an overflow or when affected by an overflow linkage event. |
| 0b1 | Monitor is captured on an overflow or when affected by an overflow linkage event. If OFLOW_FRZ is 1, the monitor does not continue to count after the overflow or overflow linkage event. If CAPT_RESET is 1, the monitor instance resets to 0. |

If RIS is implemented, this field indicates that [MSMON_CFG_CSU_CTL](#).OFLOW_CAPT is implemented for the resource instance selected by [MPAMCFG_PART_SEL](#).RIS.

Otherwise:

Reserved, RES0.

SUBTYPE, bits [22:20]

Subtype. Type of cache storage usage counted by this monitor.

This field is not currently used for CSU monitors, but reserved for future use.

This field is RAZ/WI.

Bit [19]

Reserved, RES0.

CEVNT_OFLW, bit [18]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_CSUMON_IDR.HAS_OFLOW_LNKG == 1:

Capture Event performs overflow behavior.

Selects whether a capture event matching the CAPT_EVNT field performs the overflow behavior or the capture behavior.

| CEVNT_OFLW | Meaning |
|------------|--|
| 0b0 | On a capture event matching the CAPT_EVNT field, the capture behaviors are performed. |
| 0b1 | On a capture event matching the CAPT_EVNT field, the overflow behaviors are performed. |

Otherwise:

Reserved, RES0.

MATCH_PMG, bit [17]

Match PMG.

Controls whether the monitor measures only storage used with PMG matching [MSMON_CFG_CSU_FLT.PMG](#).

| MATCH_PMG | Meaning |
|------------------|--|
| 0b0 | The monitor measures storage used with any PMG value. |
| 0b1 | The monitor only measures storage used with the PMG value matching MSMON_CFG_CSU_FLT.PMG . |

If MATCH_PMG is 1 and MATCH_PARTID is 0, it is CONSTRAINED UNPREDICTABLE whether the monitor instance:

- Measures the storage used with matching PMG and with any PARTID.
- Measures no storage usage, that is, [MSMON_CSU.VALUE](#) is zero.
- Measures the storage used with matching PMG and PARTID, that is, treats MATCH_PARTID as == 1.

MATCH_PARTID, bit [16]

Match PARTID.

Controls whether the monitor measures only storage used with PARTID matching [MSMON_CFG_CSU_FLT.PARTID](#).

| MATCH_PARTID | Meaning |
|---------------------|--|
| 0b0 | The monitor measures storage used with any PARTID value. |
| 0b1 | The monitor only measures storage used with the PARTID value matching MSMON_CFG_CSU_FLT.PARTID . |

Bits [15:11]

Reserved, RES0.

OFLOW_LNKG, bits [10:8]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_CSUMON_IDR.HAS_OFLOW_LNKG == 1:

Overflow linkage event.

Controls signaling of a capture event on overflow of this monitor instance.

| OFLOW_LNKG | Meaning |
|-------------------|--|
| 0b000 | Overflow of the monitor instance only affects this monitor instance. |
| 0b001 | Overflow of this monitor instance signals Capture Event 1. |
| 0b010 | Overflow of this monitor instance signals Capture Event 2. |
| 0b011 | Overflow of this monitor instance signals Capture Event 3. |
| 0b100 | Overflow of this monitor instance signals Capture Event 4. |
| 0b101 | Overflow of this monitor instance signals Capture Event 5. |
| 0b110 | Overflow of this monitor instance signals Capture Event 6. |
| 0b111 | Reserved. |

Otherwise:

Reserved, RES0.

TYPE, bits [7:0]

Monitor Type Code. The CSU monitor is TYPE = 0x43.

TYPE is a read-only constant indicating the type of the monitor.

Reads as 0x43.

Access to this field is **RO**.

Accessing MSMON_CFG_CSU_CTL

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CFG_CSU_CTL_s must be accessible from the Secure MPAM feature page.
- MSMON_CFG_CSU_CTL_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_CFG_CSU_CTL_rt must be accessible from the Root MPAM feature page.
- MSMON_CFG_CSU_CTL_rl must be accessible from the Realm MPAM feature page.

MSMON_CFG_CSU_CTL_s, MSMON_CFG_CSU_CTL_ns, MSMON_CFG_CSU_CTL_rt, and MSMON_CFG_CSU_CTL_rl must be separate registers:

- The Secure instance (MSMON_CFG_CSU_CTL_s) accesses the cache storage usage monitor controls used for Secure PARTIDs.
- The Non-secure instance (MSMON_CFG_CSU_CTL_ns) accesses the cache storage usage monitor controls used for Non-secure PARTIDs.
- The Root instance (MSMON_CFG_CSU_CTL_rt) accesses the cache storage usage monitor controls used for Root PARTIDs.
- The Realm instance (MSMON_CFG_CSU_CTL_rl) accesses the cache storage usage monitor controls used for Realm PARTIDs.

When RIS is implemented, loads and stores to MSMON_CFG_CSU_CTL access the cache storage usage monitor configuration settings for the cache resource instance selected by [MSMON_CFG_MON_SEL](#).RIS and the cache storage usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

When RIS is not implemented, loads and stores to MSMON_CFG_CSU_CTL access the cache storage usage monitor configuration settings for the cache storage usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

MSMON_CFG_CSU_CTL can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|---------------------|
| MPAM | MPAMF_BASE_s | 0x0818 | MSMON_CFG_CSU_CTL_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_ns | 0x0818 | MSMON_CFG_CSU_CTL_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_rt | 0x0818 | MSMON_CFG_CSU_CTL_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|-------|--------|----------|
|-----------|-------|--------|----------|

| | | | |
|------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_r1 | 0x0818 | MSMON_CFG_CSU_CTL_r1 |
|------|---------------|--------|----------------------|

When FEAT_RME is implemented access on this interface are **RW**.

MSMON_CFG_CSU_FLT, MPAM Memory System Monitor Configure Cache Storage Usage Monitor Filter Register

The MSMON_CFG_CSU_FLT characteristics are:

Purpose

Configures PARTID and PMG to measure or count in the CSU monitor selected by [MSMON_CFG_MON_SEL](#).

MSMON_CFG_CSU_FLT_s sets filter conditions for the Secure cache storage usage monitor instance selected by the Secure instance of [MSMON_CFG_MON_SEL](#). MSMON_CFG_CSU_CTL_ns sets filter conditions for the Non-secure cache storage usage monitor instance selected by the Non-secure instance of [MSMON_CFG_MON_SEL](#). MSMON_CFG_CSU_FLT_rt sets the filter conditions for the Root PARTID selected by the Root instance of [MSMON_CFG_MON_SEL](#). MSMON_CFG_CSU_FLT_rl sets the filter conditions for the Realm PARTID selected by the Realm instance of [MSMON_CFG_MON_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the monitor instance filter configuration accessed is for the resource instance currently selected by [MSMON_CFG_MON_SEL.RIS](#) and the monitor instance of that resource instance selected by [MSMON_CFG_MON_SEL.MON_SEL](#).

Configuration

The power domain of MSMON_CFG_CSU_FLT is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_CSU == 1. Otherwise, direct accesses to MSMON_CFG_CSU_FLT are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_CSU_FLT is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------------|----|----|----|----------------------|----|----|----|----|----|----|----|---------------------|----|----|----|------------------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| XCL | | | | RES0 | | | | | | | | PMG | | | | PARTID | | | | | | | | | | | | | | | |

XCL, bit [31]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_CSUMON_IDR.HAS_XCL == 1:

Exclude Clean. The monitor instance does not count cache storage used by lines in an unmodified cache state.

| XCL | Meaning |
|-----|---|
| 0b0 | Monitor instance counts cache storage in modified and unmodified cache lines. |
| 0b1 | Monitor instance counts cache storage in modified cache lines only. |

Otherwise:

Reserved, RES0.

Bits [30:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group to filter cache storage usage monitoring.

If [MSMON_CFG_CSU_CTL.MATCH_PMG](#) is 0, this field is not used to match cache storage to a PMG and the contents of this field is ignored.

If [MSMON_CFG_CSU_CTL.MATCH_PMG](#) is 1 and [MSMON_CFG_CSU_CTL.MATCH_PARTID](#) is 1, the monitor instance selected by [MSMON_CFG_MON_SEL](#) measures or counts cache storage labeled with PMG equal to this field and PARTID equal to the PARTID field.

If [MSMON_CFG_CSU_CTL.MATCH_PMG](#) is 1 and [MSMON_CFG_CSU_CTL.MATCH_PARTID](#) is 0, the behavior of the monitor instance selected by [MSMON_CFG_MON_SEL](#) is CONSTRAINED UNPREDICTABLE. See [MSMON_CFG_CSU_CTL.MATCH_PMG](#) for more information.

PARTID, bits [15:0]

Partition ID to filter cache storage usage monitoring.

If [MSMON_CFG_CSU_CTL.MATCH_PARTID](#) is 0 and [MSMON_CFG_CSU_CTL.MATCH_PMG](#) is 0, the monitor measures all allocated cache storage.

If [MSMON_CFG_CSU_CTL.MATCH_PARTID](#) is 0 and [MSMON_CFG_CSU_CTL.MATCH_PMG](#) is 1, the behavior of the monitor is CONSTRAINED UNPREDICTABLE. See the description of [MSMON_CFG_CSU_CTL.MATCH_PMG](#).

If [MSMON_CFG_CSU_CTL.MATCH_PARTID](#) is 1 and [MSMON_CFG_CSU_CTL.MATCH_PMG](#) is 0, the monitor selected by [MSMON_CFG_MON_SEL](#) measures or counts cache storage labeled with PARTID equal to this field.

If [MSMON_CFG_CSU_CTL.MATCH_PARTID](#) is 1 and [MSMON_CFG_CSU_CTL.MATCH_PMG](#) is 1, the monitor selected by [MSMON_CFG_MON_SEL](#) measures or counts cache storage labeled with PARTID equal to this field and PMG equal to the PMG field.

Accessing MSMON_CFG_CSU_FLT

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- [MSMON_CFG_CSU_FLT_s](#) must be accessible from the Secure MPAM feature page.
- [MSMON_CFG_CSU_FLT_ns](#) must be accessible from the Non-secure MPAM feature page.
- [MSMON_CFG_CSU_FLT_rt](#) must be accessible from the Root MPAM feature page.
- [MSMON_CFG_CSU_FLT_rl](#) must be accessible from the Realm MPAM feature page.

[MSMON_CFG_CSU_FLT_s](#), [MSMON_CFG_CSU_FLT_ns](#), [MSMON_CFG_CSU_FLT_rt](#), and [MSMON_CFG_CSU_FLT_rl](#) must be separate registers:

- The Secure instance ([MSMON_CFG_CSU_FLT_s](#)) accesses the PARTID and PMG matching for a cache storage usage monitor used for Secure PARTIDs.
- The Non-secure instance ([MSMON_CFG_CSU_FLT_ns](#)) accesses the PARTID and PMG matching for a cache storage usage monitor used for Non-secure PARTIDs.
- The Root instance ([MSMON_CFG_CSU_FLT_rt](#)) accesses the PARTID and PMG matching for a cache storage usage monitor used for Root PARTIDs.
- The Realm instance ([MSMON_CFG_CSU_FLT_rl](#)) accesses the PARTID and PMG matching for a cache storage usage monitor used for Realm PARTIDs.

When RIS is implemented, loads and stores to [MSMON_CFG_CSU_FLT](#) access the monitor configuration settings for the resource instance selected by [MSMON_CFG_MON_SEL.RIS](#) and the cache storage usage monitor instance selected by [MSMON_CFG_MON_SEL.MON_SEL](#).

When RIS is not implemented, loads and stores to [MSMON_CFG_CSU_FLT](#) access the monitor configuration settings for the cache storage usage monitor instance selected by [MSMON_CFG_MON_SEL.MON_SEL](#).

MSMON_CFG_CSU_FLT can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|---------------------|
| MPAM | MPAMF_BASE_s | 0x0810 | MSMON_CFG_CSU_FLT_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_ns | 0x0810 | MSMON_CFG_CSU_FLT_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_rt | 0x0810 | MSMON_CFG_CSU_FLT_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_rl | 0x0810 | MSMON_CFG_CSU_FLT_rl |

When FEAT_RME is implemented access on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_CFG_MBWU_CTL, MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Control Register

The MSMON_CFG_MBWU_CTL characteristics are:

Purpose

Controls the MBWU monitor selected by [MSMON_CFG_MON_SEL](#).

MSMON_CFG_MBWU_CTL_s controls the Secure memory bandwidth usage monitor instance selected by the Secure instance of [MSMON_CFG_MON_SEL](#). MSMON_CFG_MBWU_CTL_ns controls Non-secure memory bandwidth usage monitor instance selected by the Non-secure instance of [MSMON_CFG_MON_SEL](#). MSMON_CFG_MBWU_CTL_rt controls the monitor configuration for the Root PARTID selected by the Root instance of [MSMON_CFG_MON_SEL](#). MSMON_CFG_MBWU_CTL_rl controls the monitor configuration for the Realm PARTID selected by the Realm instance of [MSMON_CFG_MON_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the monitor instance configuration accessed is for the resource instance currently selected by [MSMON_CFG_MON_SEL](#).RIS and the monitor instance of that resource instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

Configuration

The power domain of MSMON_CFG_MBWU_CTL is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_CFG_MBWU_CTL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_MBWU_CTL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | |
|--------------------|---------------------------|----------------------------|------------------------------|----------------------------|---------------------------|----------------------------|-------------------------|-----------------------|----------------------------|--------------------|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 |
| EN | CAPT_EVNT | CAPT_RESET | OFLOW_STATUS | OFLOW_INTR | OFLOW_FRZ | OFLOW_CAPT | SUBTYPE | SCLEN | CEVNT_OFLW | MA | | | |

EN, bit [31]

Enabled.

| EN | Meaning |
|-----|--|
| 0b0 | The monitor instance is disabled and must not collect any information. |
| 0b1 | The monitor instance is enabled to collect information according to the configuration of the instance. |

CAPT_EVNT, bits [30:28]

Capture event selector.

When the selected capture event occurs, [MSMON_MBWU](#) of the monitor instance is copied to [MSMON_MBWU_CAPTURE](#) of the same instance. If the long counter is also implemented, [MSMON_MBWU_L](#) is also copied to [MSMON_MBWU_L_CAPTURE](#).

Select the event that triggers capture from the following:

| CAPT_EVNT | Meaning |
|-----------|---|
| 0b000 | No capture event is triggered. |
| 0b001 | External capture event 1 (optional, but recommended) |
| 0b010 | External capture event 2 (optional) |
| 0b011 | External capture event 3 (optional) |
| 0b100 | External capture event 4 (optional) |
| 0b101 | External capture event 5 (optional) |
| 0b110 | External capture event 6 (optional) |
| 0b111 | Capture occurs when a MSMON_CAPT_EVNT register in this MSC is written and causes a capture event for the Security state of this monitor. (optional) |

The values marked as optional indicate capture event sources that can be omitted in an implementation. Those values representing non-implemented event sources must not trigger a capture event.

If capture is not implemented for the MBWU monitor type as indicated by [MPAMF_MBWUMON_IDR](#).HAS_CAPTURE = 0, this field is RAZ/WI.

CAPT_RESET, bit [27]

Reset [MSMON_MBWU](#).VALUE after capture.

Controls whether the VALUE field of the monitor instance is reset to zero immediately after being copied to the corresponding capture register.

| CAPT_RESET | Meaning |
|------------|--|
| 0b0 | MSMON_MBWU .VALUE field of the monitor instance is not reset on capture. |
| 0b1 | MSMON_MBWU .VALUE field of the monitor instance is reset on capture. |

If capture is not implemented for the MBWU monitor type as indicated by [MPAMF_MBWUMON_IDR](#).HAS_CAPTURE = 0, this field is RAZ/WI.

This control bit affects both [MSMON_MBWU](#) and [MSMON_MBWU_L](#) in implementations that include [MSMON_MBWU_L](#).

OFLOW_STATUS, bit [26]

Overflow status.

Indicates whether the value of [MSMON_MBWU](#) has overflowed.

| OFLOW_STATUS | Meaning |
|--------------|---|
| 0b0 | MSMON_MBWU .VALUE has not overflowed. |
| 0b1 | MSMON_MBWU .VALUE has overflowed at least once since this bit was last written to zero. |

If overflow is not possible for an MBWU monitor in the MSC implementation, this field is RAZ/WI.

Overflow status for [MSMON_MBWU_L](#).VALUE is reported in [MSMON_CFG_MBWU_CTL](#).OFLOW_STATUS_L.

If [MPAMF_MBWUMON_IDR](#).HAS_OFLOW_CAPT is 1 or [MPAMF_MBWUMON_IDR](#).HAS_OFLOW_LNKG is 1, then a store to [MSMON_MBWU](#) when this field is 1 resets this field to 0.

OFLOW_INTR, bit [25]

Enable interrupt on overflow of [MSMON_MBWU](#).VALUE.

| OFLOW_INTR | Meaning |
|------------|---|
| 0b0 | No interrupt is signaled on an overflow of MSMON_MBWU .VALUE. |
| 0b1 | An implementation-specific interrupt is signaled on an overflow of MSMON_MBWU .VALUE. |

If overflow is not possible for an MBWU monitor in the MSC implementation, this field is RAZ/WI.

If overflow interrupt is not supported by the MSC implementation, this field is RAZ/WI.

Interrupt enable for overflow of [MSMON_MBWU_L](#).VALUE is controlled by [MSMON_CFG_MBWU_CTL.OFLOW_INTR_L](#).

OFLOW_FRZ, bit [24]

Freeze monitor instance on overflow.

Controls whether [MSMON_MBWU](#).VALUE field of the monitor instance freezes on an overflow.

| OFLOW_FRZ | Meaning |
|-----------|---|
| 0b0 | MSMON_MBWU .VALUE field of the monitor instance wraps on overflow. |
| 0b1 | MSMON_MBWU .VALUE field of the monitor instance freezes on overflow. If the increment that caused the overflow was 1, the frozen value is the post-increment value of 0. If the increment that caused the overflow was larger than 1, the frozen value of the monitor might be 0 or a larger value less than the final increment. |

If overflow is not possible for the instance of the MBWU monitor in the implementation, this field is RAZ/WI.

This control bit affects both [MSMON_MBWU](#) and [MSMON_MBWU_L](#) in implementations that include [MSMON_MBWU_L](#).

OFLOW_CAPT, bit [23]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_MBWUMON_IDR.HAS_OFLOW_CAPT == 1:

Capture Monitor on Overflow.

Controls whether the value of [MSMON_MBWU](#) is captured on an overflow or an overflow linkage event.

| OFLOW_CAPT | Meaning |
|------------|---|
| 0b0 | Monitor register MSMON_MBWU is not captured on an overflow or when affected by an overflow linkage event. |
| 0b1 | Monitor register MSMON_MBWU is captured on an overflow or when affected by an overflow linkage event. If OFLOW_FRZ is 1, the monitor does not continue to count after the overflow or overflow linkage event. If CAPT_RESET is 1, the monitor instance resets to 0. |

If this bit is 1, this monitor instance treats an overflow of this monitor instance as a private capture event.

If this bit is 1, this monitor instance also treats overflow linkage events for which it qualifies as a private capture event.

Otherwise:

Reserved, RES0.

SUBTYPE, bits [22:20]

Subtype. Type of bandwidth counted by this monitor.

This field is not currently used for MBWU monitors, but reserved for future use.

This field is RAZ/WI.

SCLEN, bit [19]

[MSMON_MBWU.VALUE](#) Scaling Enable.

Enables scaling of [MSMON_MBWU.VALUE](#) by [MPAMF_MBWUMON_IDR.SCALE](#).

| SCLEN | Meaning |
|-------|--|
| 0b0 | MSMON_MBWU.VALUE has bytes counted by the monitor instance. |
| 0b1 | MSMON_MBWU.VALUE has bytes counted by the monitor instance, shifted right by MPAMF_MBWUMON_IDR.SCALE . |

CEVNT_OFLW, bit [18]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_MBWUMON_IDR.HAS_OFLOW_LNKG == 1:

Capture Event performs overflow behavior.

Selects whether a capture event matching the CAPT_EVNT field perform the overflow behavior or the capture behavior.

| CEVNT_OFLW | Meaning |
|------------|---|
| 0b0 | On a capture event matching the CAPT_EVNT field the capture behaviors are performed. |
| 0b1 | On a capture event matching the CAPT_EVNT field the overflow behaviors are performed. |

Otherwise:

Reserved, RES0.

MATCH_PMG, bit [17]

Match PMG.

Controls whether the monitor instance only counts data transferred with PMG matching [MSMON_CFG_MBWU_FLT.PMG](#).

| MATCH_PMG | Meaning |
|-----------|--|
| 0b0 | The monitor instance counts data transferred with any PMG value. |
| 0b1 | The monitor instance only counts data transferred with the PMG value matching MSMON_CFG_MBWU_FLT.PMG . |

MATCH_PARTID, bit [16]

Match PARTID.

Controls whether the monitor instance counts only data transferred with PARTID matching [MSMON_CFG_MBWU_FLT.PARTID](#).

| MATCH_PARTID | Meaning |
|--------------|--|
| 0b0 | The monitor instance counts data transferred with any PARTID value. |
| 0b1 | The monitor instance only counts data transferred with the PARTID value matching MSMON_CFG_MBWU_FLT.PARTID . |

OFLOW_STATUS_L, bit [15]

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

Overflow Status of [MSMON_MBWU_L.VALUE](#) of the monitor instance.

Indicates whether [MSMON_MBWU_L.VALUE](#) has overflowed.

| OFLOW_STATUS_L | Meaning |
|----------------|---|
| 0b0 | MSMON_MBWU_L .VALUE has not overflowed. |
| 0b1 | MSMON_MBWU_L .VALUE has overflowed at least once since this bit was last written to zero. |

If [MPAMF_MBWUMON_IDR](#).HAS_LONG == 0, this bit is RES0.

Overflow status of [MSMON_MBWU](#).VALUE is reported in [MSMON_CFG_MBWU_CTL](#).OFLOW_STATUS.

If [MPAMF_MBWUMON_IDR](#).HAS_OFLOW_CAPT is 1 or [MPAMF_MBWUMON_IDR](#).HAS_OFLOW_LNKG is 1, then a store to [MSMON_MBWU_L](#) when this field is 1 resets this field to 0.

Otherwise:

Reserved, RES0.

OFLOW_INTR_L, bit [14]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and [MPAMF_MBWUMON_IDR](#).HAS_LONG == 1:

Overflow Interrupt for [MSMON_MBWU_L](#).

Controls whether an MPAM overflow interrupt is generated when [MSMON_MBWU_L](#).VALUE overflows.

| OFLOW_INTR_L | Meaning |
|--------------|--|
| 0b0 | No interrupt is signaled on an overflow of MSMON_MBWU_L .VALUE. |
| 0b1 | An implementation-specific interrupt is signaled on overflow of MSMON_MBWU_L .VALUE. |

If overflow is not possible for an MBWU monitor in the MSC implementation, this field is RAZ/WI.

If the overflow interrupt is not supported by the MSC implementation, this field is RAZ/WI.

Otherwise:

Reserved, RES0.

OFLOW_CAPT_L, bit [13]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented), [MPAMF_MBWUMON_IDR](#).HAS_LONG == 1 and [MPAMF_MBWUMON_IDR](#).HAS_OFLOW_CAPT == 1:

Capture Long Monitor on Overflow.

Controls whether [MSMON_MBWU_L](#) is captured on an overflow or an overflow linkage event.

| OFLOW_CAPT_L | Meaning |
|--------------|---|
| 0b0 | Monitor register MSMON_MBWU_L is not captured on an overflow or when affected by an overflow linkage event. |
| 0b1 | Monitor register MSMON_MBWU_L is captured on an overflow or when affected by an overflow linkage event. If OFLOW_FRZ is 1, the monitor does not continue to count after the overflow or overflow linkage event. If CAPT_RESET is 1, the monitor instance resets to 0. |

If this bit is 1, this monitor instance treats an overflow of this monitor instance as a private capture event.

If this bit is 1, this monitor instance also treats overflow linkage events for which it qualifies as a private capture event.

Otherwise:

Reserved, RES0.

Bits [12:11]

Reserved, RES0.

OFLOW_CAPT_L, bits [10:8]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented) and MPAMF_MBWUMON_IDR.HAS_OFLOW_LNKG == 1:

Overflow linkage event.

Controls signaling of a capture event on overflow of this monitor instance.

| OFLOW_CAPT_L | Meaning |
|--------------|--|
| 0b000 | Overflow of the monitor instance only affects this monitor instance. |
| 0b001 | Overflow of this monitor instance signals Capture Event 1. |
| 0b010 | Overflow of this monitor instance signals Capture Event 2. |
| 0b011 | Overflow of this monitor instance signals Capture Event 3. |
| 0b100 | Overflow of this monitor instance signals Capture Event 4. |
| 0b101 | Overflow of this monitor instance signals Capture Event 5. |
| 0b110 | Overflow of this monitor instance signals Capture Event 6. |
| 0b111 | Reserved. |

Otherwise:

Reserved, RES0.

TYPE, bits [7:0]

Monitor Type Code. The MBWU monitor is TYPE = 0x42.

TYPE is a read-only constant indicating the type of the monitor.

Reads as 0x42.

Access to this field is **RO**.

Accessing MSMON_CFG_MBWU_CTL

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CFG_MBWU_CTL_s must be accessible from the Secure MPAM feature page.
- MSMON_CFG_MBWU_CTL_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_CFG_MBWU_CTL_rt must be accessible from the Root MPAM feature page.
- MSMON_CFG_MBWU_CTL_rl must be accessible from the Realm MPAM feature page.

MSMON_CFG_MBWU_CTL_s, MSMON_CFG_MBWU_CTL_ns, MSMON_CFG_MBWU_CTL_rt, and MSMON_CFG_MBWU_CTL_rl must be separate registers:

- The Secure instance (MSMON_CFG_MBWU_CTL_s) accesses the memory bandwidth usage monitor controls used for Secure PARTIDs.

- The Non-secure instance (MSMON_CFG_MBWU_CTL_ns) accesses the memory bandwidth usage monitor controls used for Non-secure PARTIDs.
- The Root instance (MSMON_CFG_MBWU_CTL_rt) accesses the memory bandwidth usage monitor controls used for Root PARTIDs.
- The Realm instance (MSMON_CFG_MBWU_CTL_rl) accesses the memory bandwidth usage monitor controls used for Realm PARTIDs.

When RIS is implemented, loads and stores to MSMON_CFG_MBWU_CTL access the monitor configuration settings for the bandwidth resource instance selected by [MSMON_CFG_MON_SEL](#).RIS and the memory bandwidth usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

When RIS is not implemented, loads and stores to MSMON_CFG_MBWU_CTL access the monitor configuration settings for the memory bandwidth usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

MSMON_CFG_MBWU_CTL can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|----------------------|
| MPAM | MPAMF_BASE_s | 0x0828 | MSMON_CFG_MBWU_CTL_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_ns | 0x0828 | MSMON_CFG_MBWU_CTL_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rt | 0x0828 | MSMON_CFG_MBWU_CTL_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rl | 0x0828 | MSMON_CFG_MBWU_CTL_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MSMON_CFG_MBWU_FLT, MPAM Memory System Monitor Configure Memory Bandwidth Usage Monitor Filter Register

The MSMON_CFG_MBWU_FLT characteristics are:

Purpose

Controls PARTID and PMG to measure or count in the MBWU monitor selected by [MSMON_CFG_MON_SEL](#).

MSMON_CFG_MBWU_FLT_s sets filter conditions for the Secure memory bandwidth usage monitor instance selected by the Secure instance of [MSMON_CFG_MON_SEL](#). MSMON_CFG_MBWU_CTL_ns sets filter conditions for the Non-secure memory bandwidth usage monitor instance selected by the Non-secure instance of [MSMON_CFG_MON_SEL](#). MSMON_CFG_CSU_FLT_rt sets the filter conditions for the Root PARTID selected by the Root instance of [MSMON_CFG_MON_SEL](#). MSMON_CFG_CSU_FLT_rl sets the filter conditions for the Realm PARTID selected by the Realm instance of [MSMON_CFG_MON_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the monitor instance filter configuration accessed is for the resource instance currently selected by [MSMON_CFG_MON_SEL.RIS](#) and the monitor instance of that resource instance selected by [MSMON_CFG_MON_SEL.MON_SEL](#).

Configuration

The power domain of MSMON_CFG_MBWU_FLT is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_CFG_MBWU_FLT are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_MBWU_FLT is a 32-bit register.

Field descriptions

When FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|--------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RWBW | | RES0 | | | | | | | | PMG | | | | | | | | PARTID | | | | | | | | | | | | | |

RW filtering.

RWBW, bits [31:30]

When [MPAMF_MBWUMON_IDR.HAS_RWBW](#) == 1:

Read/write bandwidth filter. Configures the selected monitor instance to count all bandwidth, only read bandwidth or only write bandwidth.

| RWBW | Meaning |
|------|---|
| 0b00 | Monitor instance counts read bandwidth and write bandwidth. |
| 0b01 | Monitor instance counts write bandwidth only. |
| 0b10 | Monitor instance counts read bandwidth only. |
| 0b11 | Reserved. |

Otherwise:

Reserved, RES0.

Bits [29:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group to filter memory bandwidth usage monitoring.

If [MSMON_CFG_MBWU_CTL.MATCH_PMG](#) == 0, this field is not used to match memory bandwidth to a PMG and the contents of this field is ignored.

If [MSMON_CFG_MBWU_CTL.MATCH_PMG](#) == 1, the monitor selected by [MSMON_CFG_MON_SEL](#) measures or counts memory bandwidth labeled with PMG equal to this field.

PARTID, bits [15:0]

Partition ID to filter memory bandwidth usage monitoring.

If [MSMON_CFG_MBWU_CTL.MATCH_PARTID](#) == 0, this field is not used to match memory bandwidth to a PARTID and the contents of this field is ignored.

If [MSMON_CFG_MBWU_CTL.MATCH_PARTID](#) == 1, the monitor selected by [MSMON_CFG_MON_SEL](#) measures or counts memory bandwidth labeled with PARTID equal to this field.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | PMG | | | | | | | | PARTID | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group to filter memory bandwidth usage monitoring.

If [MSMON_CFG_MBWU_CTL.MATCH_PMG](#) == 0, this field is not used to match memory bandwidth to a PMG and the contents of this field is ignored.

If [MSMON_CFG_MBWU_CTL.MATCH_PMG](#) == 1, the monitor selected by [MSMON_CFG_MON_SEL](#) measures or counts memory bandwidth labeled with PMG equal to this field.

PARTID, bits [15:0]

Partition ID to filter memory bandwidth usage monitoring.

If [MSMON_CFG_MBWU_CTL.MATCH_PARTID](#) == 0, this field is not used to match memory bandwidth to a PARTID and the contents of this field is ignored.

If [MSMON_CFG_MBWU_CTL.MATCH_PARTID](#) == 1, the monitor selected by [MSMON_CFG_MON_SEL](#) measures or counts memory bandwidth labeled with PARTID equal to this field.

Accessing MSMON_CFG_MBWU_FLT

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CFG_MBWU_FLT_s must be accessible from the Secure MPAM feature page.
- MSMON_CFG_MBWU_FLT_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_CFG_MBWU_FLT_rt must be accessible from the Root MPAM feature page.
- MSMON_CFG_MBWU_FLT_rl must be accessible from the Realm MPAM feature page.

MSMON_CFG_MBWU_FLT_s, MSMON_CFG_MBWU_FLT_ns, MSMON_CFG_MBWU_FLT_rt, and MSMON_CFG_MBWU_FLT_rl must be separate registers:

- The Secure instance (MSMON_CFG_MBWU_FLT_s) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_CFG_MBWU_FLT_ns) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_CFG_MBWU_FLT_rt) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_CFG_MBWU_FLT_rl) accesses the PARTID and PMG matching for a memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, loads and stores to MSMON_CFG_MBWU_FLT access the monitor configuration settings for the bandwidth resource instance selected by [MSMON_CFG_MON_SEL](#).RIS and the memory bandwidth usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

When RIS is not implemented, loads and stores to MSMON_CFG_MBWU_FLT access the monitor configuration settings for the memory bandwidth usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

MSMON_CFG_MBWU_FLT can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|----------------------|
| MPAM | MPAMF_BASE_s | 0x0820 | MSMON_CFG_MBWU_FLT_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_ns | 0x0820 | MSMON_CFG_MBWU_FLT_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rt | 0x0820 | MSMON_CFG_MBWU_FLT_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rl | 0x0820 | MSMON_CFG_MBWU_FLT_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MSMON_CFG_MON_SEL, MPAM Monitor Instance Selection Register

The MSMON_CFG_MON_SEL characteristics are:

Purpose

Selects a monitor instance to access through the MSMON configuration and counter registers.

MSMON_CFG_MON_SEL_s selects a Secure monitor instance to access via the Secure MPAM feature page.
 MSMON_CFG_MON_SEL_ns selects a Non-secure monitor instance to access via the Non-secure MPAM feature page.
 MSMON_CFG_MON_SEL_rt selects a Root monitor instance to access via the Root MPAM feature page.
 MSMON_CFG_MON_SEL_rl selects a Realm monitor instance to access via the Realm MPAM feature page.

Note

Different performance monitoring features within an MSC could have different numbers of monitor instances. See the NUM_MON field in the corresponding ID register. This means that a monitor out-of-bounds error might be signaled when an MSMON_CFG register is accessed because the value in MSMON_CFG_MON_SEL.MON_SEL is too large for the particular monitoring feature.

To configure a monitor, set MON_SEL in this register to the index of the monitor instance to configure, then write to the MSMON_CFG_x register to set the configuration of the monitor. At a later time, read the monitor register (for example, MSMON_MBWU) to get the value of the monitor.

Configuration

The power domain of MSMON_CFG_MON_SEL is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented and (MPAMF_IDR.HAS_MSMON == 1, or (MPAMF_IDR.HAS_IMPL_IDR == 1 and MPAMF_IDR.EXT == 0) or (MPAMF_IDR.HAS_IMPL_IDR == 1, MPAMF_IDR.EXT == 1 and MPAMF_IDR.NO_IMPL_MSMON == 0)). Otherwise, direct accesses to MSMON_CFG_MON_SEL are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CFG_MON_SEL is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-----|----|----|----|------|----|----|----|----|----|----|----|---------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | RIS | | | | RES0 | | | | | | | | MON_SEL | | | | | | | | | | | | | | | |

Bits [31:28]

Reserved, RES0.

RIS, bits [27:24]

When (FEAT_MPAMv0p1 is implemented or FEAT_MPAMv1p1 is implemented), MPAMF_IDR.EXT == 1 and MPAMF_IDR.HAS_RIS == 1:

Resource Instance Selector. RIS selects one resource to configure through MSMON_CFG registers.

Otherwise:

Reserved, RES0.

Bits [23:16]

Reserved, RES0.

MON_SEL, bits [15:0]

Selects the monitor instance to configure or read.

Reads and writes to other MSMON registers are indexed by MON_SEL and by the NS bit used to access MSMON_CFG_MON_SEL to access the configuration for a single monitor.

Accessing MSMON_CFG_MON_SEL

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CFG_MON_SEL_s must be accessible from the Secure MPAM feature page.
- MSMON_CFG_MON_SEL_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_CFG_MON_SEL_rt must be accessible from the Root MPAM feature page.
- MSMON_CFG_MON_SEL_rl must be accessible from the Realm MPAM feature page.

MSMON_CFG_MON_SEL_s, MSMON_CFG_MON_SEL_ns, MSMON_CFG_MON_SEL_rt, and MSMON_CFG_MON_SEL_rl must be separate registers:

- The Secure instance (MSMON_CFG_MON_SEL_s) accesses the monitor instance selector used for Secure PARTIDs.
- The Non-secure instance (MSMON_CFG_MON_SEL_ns) accesses the monitor instance selector used for Non-secure PARTIDs.
- The Root instance (MSMON_CFG_MON_SEL_rt) accesses the monitor instance selector used for Root PARTIDs.
- The Realm instance (MSMON_CFG_MON_SEL_rl) accesses the monitor instance selector used for Realm PARTIDs.

MSMON_CFG_MON_SEL can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|---------------------|
| MPAM | MPAMF_BASE_s | 0x0800 | MSMON_CFG_MON_SEL_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_ns | 0x0800 | MSMON_CFG_MON_SEL_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_rt | 0x0800 | MSMON_CFG_MON_SEL_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_rl | 0x0800 | MSMON_CFG_MON_SEL_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MSMON_CSU, MPAM Cache Storage Usage Monitor Register

The MSMON_CSU characteristics are:

Purpose

Accesses the CSU monitor instance selected by [MSMON_CFG_MON_SEL](#).

MSMON_CSU_s is a Secure cache storage usage monitor instance selected by the Secure instance of [MSMON_CFG_MON_SEL](#). MSMON_CSU_ns is a Non-secure cache storage usage monitor instance selected by the Non-secure instance of [MSMON_CFG_MON_SEL](#). MSMON_CSU_rt is a Root cache storage usage monitor instance selected by the Root instance of [MSMON_CFG_MON_SEL](#). MSMON_CSU_rl is a Realm cache storage usage monitor instance selected by the Realm instance of [MSMON_CFG_MON_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the monitor instance accessed is for the resource instance currently selected by [MSMON_CFG_MON_SEL.RIS](#) and the monitor instance of that resource instance selected by [MSMON_CFG_MON_SEL.MON_SEL](#).

Configuration

The power domain of MSMON_CSU is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, [MPAMF_IDR.HAS_MSMON](#) == 1 and [MPAMF_MSMON_IDR.MSMON_CSU](#) == 1. Otherwise, direct accesses to MSMON_CSU are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CSU is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NRDY | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |

NRDY, bit [31]

Not Ready. Indicates whether the monitor instance has possibly inaccurate data.

| NRDY | Meaning |
|------|--|
| 0b0 | The monitor instance is ready and the MSMON_CSU.VALUE field is accurate. |
| 0b1 | The monitor instance is not ready and the contents of the MSMON_CSU.VALUE field might be inaccurate or otherwise not represent the actual cache storage usage. |

VALUE, bits [30:0]

Cache storage usage measurement value if MSMON_CSU.NRDY is 0. Invalid if MSMON_CSU.NRDY is 1.

VALUE is the cache storage usage measured in bytes meeting the criteria set in [MSMON_CFG_CSU_FLT](#) and [MSMON_CFG_CSU_CTL](#) for the monitor instance selected by [MSMON_CFG_MON_SEL](#).

Accessing MSMON_CSU

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CSU_s must be accessible from the Secure MPAM feature page.
- MSMON_CSU_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_CSU_rt must be accessible from the Root MPAM feature page.
- MSMON_CSU_rl must be accessible from the Realm MPAM feature page.

MSMON_CSU_s, MSMON_CSU_ns, MSMON_CSU_rt, and MSMON_CSU_rl must be separate registers:

- The Secure instance (MSMON_CSU_s) accesses the cache storage usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_CSU_ns) accesses the cache storage usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_CSU_rt) accesses the cache storage usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_CSU_rl) accesses the cache storage usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_CSU access the cache storage usage monitor monitor instance for the cache resource instance selected by [MSMON_CFG_MON_SEL](#).RIS and the cache storage usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

When RIS is not implemented, reads and writes to MSMON_CSU access the cache storage usage monitor monitor instance for the cache storage usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

MSMON_CSU can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|-------------|
| MPAM | MPAMF_BASE_s | 0x0840 | MSMON_CSU_s |

This interface is accessible as follows:

- When MPAMF_CSUMON_IDR.CSU_RO == 0 accesses to this register are **RW**.
- When MPAMF_CSUMON_IDR.CSU_RO == 1 accesses to this register are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------|
| MPAM | MPAMF_BASE_ns | 0x0840 | MSMON_CSU_ns |

This interface is accessible as follows:

- When MPAMF_CSUMON_IDR.CSU_RO == 0 accesses to this register are **RW**.
- When MPAMF_CSUMON_IDR.CSU_RO == 1 accesses to this register are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------|
| MPAM | MPAMF_BASE_rt | 0x0840 | MSMON_CSU_rt |

This interface is accessible as follows:

- When FEAT_RME is implemented and MPAMF_CSUMON_IDR.CSU_RO == 0 accesses to this register are **RW**.
- When FEAT_RME is implemented and MPAMF_CSUMON_IDR.CSU_RO == 1 accesses to this register are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------|
| MPAM | MPAMF_BASE_rl | 0x0840 | MSMON_CSU_rl |

This interface is accessible as follows:

- When FEAT_RME is implemented and MPAMF_CSUMON_IDR.CSU_RO == 0 accesses to this register are **RW**.
- When FEAT_RME is implemented and MPAMF_CSUMON_IDR.CSU_RO == 1 accesses to this register are **RO**.

MSMON_CSU_CAPTURE, MPAM Cache Storage Usage Monitor Capture Register

The MSMON_CSU_CAPTURE characteristics are:

Purpose

MSMON_CSU_CAPTURE is a 32-bit read-write register that accesses the captured [MSMON_CSU](#) monitor instance selected by [MSMON_CFG_MON_SEL](#).

MSMON_CSU_CAPTURE_s is the Secure cache storage usage monitor capture instance selected by the Secure instance of [MSMON_CFG_MON_SEL](#). MSMON_CSU_CAPTURE_ns is the Non-secure cache storage usage monitor capture instance selected by the Non-secure instance of [MSMON_CFG_MON_SEL](#). MSMON_CSU_CAPTURE_rt is a Root cache storage usage monitor capture instance selected by the Root instance of [MSMON_CFG_MON_SEL](#). MSMON_CSU_CAPTURE_rl is a Realm cache storage usage monitor capture instance selected by the Realm instance of [MSMON_CFG_MON_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the monitor instance capture register accessed is for the resource instance currently selected by [MSMON_CFG_MON_SEL.RIS](#) and the monitor instance of that resource instance selected by [MSMON_CFG_MON_SEL.MON_SEL](#).

Configuration

The power domain of MSMON_CSU_CAPTURE is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_CSU == 1 and MPAMF_CSUMON_IDR.HAS_CAPTURE == 1. Otherwise, direct accesses to MSMON_CSU_CAPTURE are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CSU_CAPTURE is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NRDY | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |

NRDY, bit [31]

Not Ready. Indicates whether the captured monitor value has possibly inaccurate data.

| NRDY | Meaning |
|------|--|
| 0b0 | The captured monitor instance was ready and the MSMON_CSU_CAPTURE.VALUE field is accurate. |
| 0b1 | The captured monitor instance was not ready and the contents of the MSMON_CSU_CAPTURE.VALUE field might be inaccurate or otherwise not represent the actual cache storage usage. |

VALUE, bits [30:0]

Captured cache storage usage measurement if MSMON_CSU_CAPTURE.NRDY is 0. Invalid if MSMON_CSU_CAPTURE.NRDY is 1.

VALUE is the captured cache storage usage measurement in bytes meeting the criteria set in [MSMON_CFG_CSU_FLT](#) and [MSMON_CFG_CSU_CTL](#) for the monitor instance selected by [MSMON_CFG_MON_SEL](#).

Accessing MSMON_CSU_CAPTURE

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- `MSMON_CSU_CAPTURE_s` must be accessible from the Secure MPAM feature page.
- `MSMON_CSU_CAPTURE_ns` must be accessible from the Non-secure MPAM feature page.
- `MSMON_CSU_CAPTURE_rt` must be accessible from the Root MPAM feature page.
- `MSMON_CSU_CAPTURE_rl` must be accessible from the Realm MPAM feature page.

`MSMON_CSU_CAPTURE_s`, `MSMON_CSU_CAPTURE_ns`, `MSMON_CSU_CAPTURE_rt`, and `MSMON_CSU_CAPTURE_rl` must be separate registers:

- The Secure instance (`MSMON_CSU_CAPTURE_s`) accesses the captured cache storage usage monitor used for Secure PARTIDs.
- The Non-secure instance (`MSMON_CSU_CAPTURE_ns`) accesses the captured cache storage usage monitor used for Non-secure PARTIDs.
- The Root instance (`MSMON_CSU_CAPTURE_rt`) accesses the captured cache storage usage monitor used for Root PARTIDs.
- The Realm instance (`MSMON_CSU_CAPTURE_rl`) accesses the captured cache storage usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to `MSMON_CSU_CAPTURE` access the monitor instance for the cache resource instance selected by [MSMON_CFG_MON_SEL](#).RIS and the cache storage usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

When RIS is not implemented, reads and writes to `MSMON_CSU_CAPTURE` access the monitor instance for the cache storage usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

MSMON_CSU_CAPTURE can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|---------------------|
| MPAM | MPAMF_BASE_s | 0x0848 | MSMON_CSU_CAPTURE_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_ns | 0x0848 | MSMON_CSU_CAPTURE_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_rt | 0x0848 | MSMON_CSU_CAPTURE_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|----------------------|
| MPAM | MPAMF_BASE_rl | 0x0848 | MSMON_CSU_CAPTURE_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MSMON_CSU_OFSR, MPAM CSU Monitor Overflow Status Register

The MSMON_CSU_OFSR characteristics are:

Purpose

MSMON_CSU_OFSR is a 32-bit read-only register that shows bitmap of CSU monitor instance overflow status for a contiguous group of 32 monitor instances.

MSMON_CSU_OFSR_s gives a bitmap of pending CSU overflow status for 32 Secure CSU monitor instances.
 MSMON_CSU_OFSR_ns gives a bitmap of pending CSU overflow status for 32 Non-secure CSU monitor instances.
 MSMON_CSU_OFSR_rt gives a bitmap of pending CSU overflow status for 32 Root CSU monitor instances.
 MSMON_CSU_OFSR_rl gives a bitmap of pending CSU overflow status for 32 Realm CSU monitor instances.

Configuration

The power domain of MSMON_CSU_OFSR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_CSUMON_IDR.HAS_OFSR == 1. Otherwise, direct accesses to MSMON_CSU_OFSR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_CSU_OFSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| OFPND31 | OFPND30 | OFPND29 | OFPND28 | OFPND27 | OFPND26 | OFPND25 | OFPND24 | OFPND23 | OFPND22 | OFPND21 | OFPND20 |

OFPND<i>, bit [i], for i = 31 to 0

Overflow status bitmap for CSU monitor instances. The RIS and the contiguous range of CSU monitor instances are set in [MSMON_CFG_MON_SEL](#). i of 0 corresponds to the CSU monitor instance [MSMON_CFG_MON_SEL.MON_SEL & 0xFFE0](#).

| OFPND<i> | Meaning |
|----------|---|
| 0b0 | CSU monitor instance (MSMON_CFG_MON_SEL.MON_SEL & 0xFFE0 + i) does not have a pending overflow. |
| 0b1 | CSU monitor instance (MSMON_CFG_MON_SEL.MON_SEL & 0xFFE0 + i) has a pending overflow. |

After reading [MSMON_OFLOW_SR](#) to determine that a CSU monitor instance has a pending overflow and which RIS values have pending overflows, an interrupt service routine could poll groups of 32 monitor instances in a RIS for pending monitors by reading this bitmap and incrementing [MSMON_CFG_MON_SEL.MON_SEL](#) by 32.

Accessing MSMON_CSU_OFSR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_CSU_OFSR_s must be accessible from the Secure MPAM feature page.
- MSMON_CSU_OFSR_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_CSU_OFSR_rt must be accessible from the Root MPAM feature page.
- MSMON_CSU_OFSR_rl must be accessible from the Realm MPAM feature page.

MSMON_CSU_OFSR_s, MSMON_CSU_OFSR_ns, MSMON_CSU_OFSR_rt, and MSMON_CSU_OFSR_rl must be separate registers:

- The Secure instance (MSMON_CSU_OFSR_s) accesses the CSU monitor overflow status bitmap used for Secure PARTIDs.
- The Non-secure instance (MSMON_CSU_OFSR_ns) accesses the CSU monitor overflow status bitmap used for Non-secure PARTIDs.
- The Root instance (MSMON_CSU_OFSR_rt) accesses the CSU monitor overflow status bitmap used for Root PARTIDs.
- The Realm instance (MSMON_CSU_OFSR_rl) accesses the CSU monitor overflow status bitmap used for Realm PARTIDs.

MSMON_CSU_OFSR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|------------------|
| MPAM | MPAMF_BASE_s | 0x0858 | MSMON_CSU_OFSR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_ns | 0x0858 | MSMON_CSU_OFSR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_rt | 0x0858 | MSMON_CSU_OFSR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_rl | 0x0858 | MSMON_CSU_OFSR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

MSMON_MBWU, MPAM Memory Bandwidth Usage Monitor Register

The MSMON_MBWU characteristics are:

Purpose

Accesses the monitor instance selected by [MSMON_CFG_MON_SEL](#).

MSMON_MBWU_s is the Secure memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_s. MSMON_MBWU_ns is the Non-secure memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_ns. MSMON_MBWU_rt is the Root memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_rt. MSMON_MBWU_rl is the Realm memory bandwidth usage monitor instance selected by MSMON_CFG_MON_SEL_rl.

If [MPAMF_IDR.HAS_RIS](#) is 1, the monitor instance register accessed is for the resource instance currently selected by [MSMON_CFG_MON_SEL](#).RIS and the monitor instance of that resource instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

Configuration

The power domain of MSMON_MBWU is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1 and MPAMF_MSMON_IDR.MSMON_MBWU == 1. Otherwise, direct accesses to MSMON_MBWU are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NRDY | | VALUE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

NRDY, bit [31]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

| NRDY | Meaning |
|------|--|
| 0b0 | The monitor instance is ready and the MSMON_MBWU.VALUE field is accurate. |
| 0b1 | The monitor instance is not ready and the contents of the MSMON_MBWU.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage. |

VALUE, bits [30:0]

Memory bandwidth usage counter value if MSMON_MBWU.NRDY is 0. Invalid if MSMON_MBWU.NRDY is 1.

VALUE is the scaled count of bytes transferred since the monitor was last reset that met the criteria set in [MSMON_CFG_MBWU_FLT](#) and [MSMON_CFG_MBWU_CTL](#) for the monitor instance selected by [MSMON_CFG_MON_SEL](#).

If [MSMON_CFG_MBWU_CTL](#).SCLEN enables scaling, the count in VALUE is the number of bytes shifted right by [MPAMF_MBWUMON_IDR](#).SCALE bit positions and rounded.

Accessing MSMON_MBWU

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_MBWU_s must be accessible from the Secure MPAM feature page.
- MSMON_MBWU_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_MBWU_rt must be accessible from the Root MPAM feature page.
- MSMON_MBWU_rl must be accessible from the Realm MPAM feature page.

MSMON_MBWU_s, MSMON_MBWU_ns, MSMON_MBWU_rt, and MSMON_MBWU_rl must be separate registers:

- The Secure instance (MSMON_MBWU_s) accesses the memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_ns) accesses the memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_rt) accesses the memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_MBWU_rl) accesses the memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_MBWU access the memory bandwidth usage monitor instance for the resource instance selected by [MSMON_CFG_MON_SEL](#). RIS and the memory bandwidth usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

When RIS is not implemented, reads and writes to MSMON_MBWU access the memory bandwidth usage monitor instance for the memory bandwidth usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

MSMON_MBWU can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|--------------|
| MPAM | MPAMF_BASE_s | 0x0860 | MSMON_MBWU_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_ns | 0x0860 | MSMON_MBWU_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_rt | 0x0860 | MSMON_MBWU_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_rl | 0x0860 | MSMON_MBWU_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MSMON_MBWU_CAPTURE, MPAM Memory Bandwidth Usage Monitor Capture Register

The MSMON_MBWU_CAPTURE characteristics are:

Purpose

Accesses the captured MSMON_MBWU monitor instance selected by [MSMON_CFG_MON_SEL](#).

MSMON_MBWU_CAPTURE_s is the Secure memory bandwidth usage monitor capture instance selected by the Secure instance of [MSMON_CFG_MON_SEL](#). MSMON_MBWU_CAPTURE_ns is the Non-secure memory bandwidth usage monitor capture instance selected by the Non-secure instance of [MSMON_CFG_MON_SEL](#).

MSMON_MBWU_CAPTURE_rt is the Root memory bandwidth usage monitor capture instance selected by the Root instance of [MSMON_CFG_MON_SEL](#). MSMON_MBWU_CAPTURE_rl is the Realm memory bandwidth usage monitor capture instance selected by the Realm instance of [MSMON_CFG_MON_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the monitor instance capture register accessed is for the resource instance currently selected by [MSMON_CFG_MON_SEL.RIS](#) and the monitor instance of that resource instance selected by [MSMON_CFG_MON_SEL.MON_SEL](#).

Configuration

The power domain of MSMON_MBWU_CAPTURE is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1 and MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1. Otherwise, direct accesses to MSMON_MBWU_CAPTURE are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU_CAPTURE is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NRDY | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |

NRDY, bit [31]

Not Ready. The captured NRDY bit from the corresponding instance of [MSMON_MBWU](#). This bit indicates whether the captured monitor value has possibly inaccurate data.

| NRDY | Meaning |
|------|--|
| 0b0 | The captured monitor instance was ready and the MSMON_MBWU_CAPTURE.VALUE field is accurate. |
| 0b1 | The captured monitor instance was not ready and the contents of the MSMON_MBWU_CAPTURE.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage. |

VALUE, bits [30:0]

Captured memory bandwidth usage counter value if MSMON_MBWU_CAPTURE.NRDY is 0. Invalid if MSMON_MBWU_CAPTURE.NRDY is 1.

VALUE is the captured VALUE field from the corresponding instance of [MSMON_MBWU](#), the count of bytes transferred since the monitor was last reset that meet the criteria set in [MSMON_CFG_MBWU_FLT](#) and [MSMON_CFG_MBWU_CTL](#) for the monitor instance selected by [MSMON_CFG_MON_SEL](#).

VALUE captures the [MSMON_MBWU](#).VALUE and preserves any scaling that had been performed on the VALUE field in that register.

Accessing MSMON_MBWU_CAPTURE

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- [MSMON_MBWU_CAPTURE_s](#) must be accessible from the Secure MPAM feature page.
- [MSMON_MBWU_CAPTURE_ns](#) must be accessible from the Non-secure MPAM feature page.
- [MSMON_MBWU_CAPTURE_rt](#) must be accessible from the Root MPAM feature page.
- [MSMON_MBWU_CAPTURE_rl](#) must be accessible from the Realm MPAM feature page.

[MSMON_MBWU_CAPTURE_s](#), [MSMON_MBWU_CAPTURE_ns](#), [MSMON_MBWU_CAPTURE_rt](#), and [MSMON_MBWU_CAPTURE_rl](#) must be separate registers:

- The Secure instance ([MSMON_MBWU_CAPTURE_s](#)) accesses the captured memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance ([MSMON_MBWU_CAPTURE_ns](#)) accesses the captured memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance ([MSMON_MBWU_CAPTURE_rt](#)) accesses the captured memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance ([MSMON_MBWU_CAPTURE_rl](#)) accesses the captured memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to [MSMON_MBWU_CAPTURE](#) access the monitor instance for the bandwidth resource instance selected by [MSMON_CFG_MON_SEL](#).RIS and the memory bandwidth usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

When RIS is not implemented, reads and writes to [MSMON_MBWU_CAPTURE](#) access the monitor instance for the memory bandwidth usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

MSMON_MBWU_CAPTURE can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|----------------------|
| MPAM | MPAMF_BASE_s | 0x0868 | MSMON_MBWU_CAPTURE_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_ns | 0x0868 | MSMON_MBWU_CAPTURE_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rt | 0x0868 | MSMON_MBWU_CAPTURE_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rl | 0x0868 | MSMON_MBWU_CAPTURE_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MSMON_MBWU_L, MPAM Long Memory Bandwidth Usage Monitor Register

The MSMON_MBWU_L characteristics are:

Purpose

Accesses the monitor instance selected by [MSMON_CFG_MON_SEL](#).

MSMON_MBWU_L_s is the Secure long memory bandwidth usage monitor instance selected by the Secure instance of [MSMON_CFG_MON_SEL](#). MSMON_MBWU_L_ns is the Non-secure long memory bandwidth usage monitor instance selected by the Non-secure instance of [MSMON_CFG_MON_SEL](#). MSMON_MBWU_L_rt is the Root long memory bandwidth usage monitor instance selected by [MSMON_CFG_MON_SEL](#)_rt. MSMON_MBWU_L_rl is the Realm long memory bandwidth usage monitor instance selected by [MSMON_CFG_MON_SEL](#)_rl.

If [MPAMF_IDR](#).HAS_RIS is 1, the monitor instance long monitor register accessed is for the resource instance currently selected by [MSMON_CFG_MON_SEL](#).RIS and the monitor instance of that resource instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

Configuration

The power domain of MSMON_MBWU_L is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1 and MPAMF_MBWUMON_IDR.HAS_LONG == 1. Otherwise, direct accesses to MSMON_MBWU_L are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU_L is a 64-bit register.

Field descriptions

When MPAMF_MBWUMON_IDR.LWD == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| NRDY | RES0 | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

NRDY, bit [63]

Not Ready. Indicates whether the monitor instance has possibly inaccurate data.

| NRDY | Meaning |
|------|--|
| 0b0 | The monitor instance is ready and the MSMON_MBWU_L.VALUE field is accurate. |
| 0b1 | The monitor instance is not ready and the contents of the MSMON_MBWU_L.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage. |

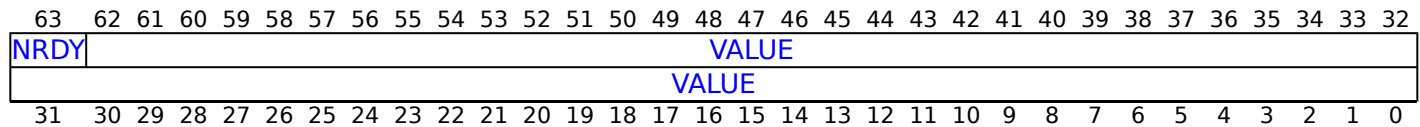
Bits [62:44]

Reserved, RES0.

VALUE, bits [43:0]

Long (44-bit) memory bandwidth usage counter value if MSMON_MBWU_L.NRDY is 0. Invalid if MSMON_MBWU_L.NRDY is 1.

VALUE is the long count of bytes transferred since the monitor was last reset that met the criteria set in [MSMON_CFG_MBWU_FLT](#) and [MSMON_CFG_MBWU_CTL](#) for the monitor instance selected by [MSMON_CFG_MON_SEL](#).

When MPAMF_MBWUMON_IDR.LWD == 1:**NRDY, bit [63]**

Not Ready. Indicates whether the monitor instance has possibly inaccurate data.

| NRDY | Meaning |
|------|--|
| 0b0 | The monitor instance is ready and the MSMON_MBWU_L.VALUE field is accurate. |
| 0b1 | The monitor instance is not ready and the contents of the MSMON_MBWU_L.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage. |

VALUE, bits [62:0]

Long (63-bit) memory bandwidth usage counter value if MSMON_MBWU_L.NRDY is 0. Invalid if MSMON_MBWU_L.NRDY is 1.

VALUE is the long count of bytes transferred since the monitor instance was last reset that met the criteria set in [MSMON_CFG_MBWU_FLT](#) and [MSMON_CFG_MBWU_CTL](#) for the monitor instance selected by [MSMON_CFG_MON_SEL](#).

Accessing MSMON_MBWU_L

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_MBWU_L_s must be accessible from the Secure MPAM feature page.
- MSMON_MBWU_L_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_MBWU_L_rt must be accessible from the Root MPAM feature page.
- MSMON_MBWU_L_rl must be accessible from the Realm MPAM feature page.

MSMON_MBWU_L_s, MSMON_MBWU_L_ns, MSMON_MBWU_L_rt, and MSMON_MBWU_L_rl must be separate registers:

- The Secure instance (MSMON_MBWU_L_s) accesses the long memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_L_ns) accesses the long memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_L_rt) accesses the long memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_MBWU_L_rl) accesses the long memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_MBWU_L access the long memory bandwidth usage monitor instance for the bandwidth resource instance selected by [MSMON_CFG_MON_SEL](#).RIS and the monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

When RIS is not implemented, reads and writes to MSMON_MBWU_L access the long memory bandwidth usage monitor instance for the monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

MSMON_MBWU_L can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|--------------|
| MPAM | MPAMF_BASE_s | 0x0880 | MSMON_MBWU_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_ns | 0x0880 | MSMON_MBWU_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_rt | 0x0880 | MSMON_MBWU_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------|
| MPAM | MPAMF_BASE_rl | 0x0880 | MSMON_MBWU_rl |

When FEAT_RME is implemented access on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_MBWU_L_CAPTURE, MPAM Long Memory Bandwidth Usage Monitor Capture Register

The MSMON_MBWU_L_CAPTURE characteristics are:

Purpose

Accesses the captured [MSMON_MBWU_L](#) monitor instance selected by [MSMON_CFG_MON_SEL](#).

MSMON_MBWU_L_CAPTURE_s is the Secure long memory bandwidth usage monitor capture instance selected by the Secure instance of [MSMON_CFG_MON_SEL](#). MSMON_MBWU_L_CAPTURE_ns is the Non-secure long memory bandwidth usage monitor capture instance selected by the Non-secure instance of [MSMON_CFG_MON_SEL](#). MSMON_MBWU_L_CAPTURE_rt is the Root long memory bandwidth usage monitor capture instance selected by the Root instance of [MSMON_CFG_MON_SEL](#). MSMON_MBWU_L_CAPTURE_rl is the Realm long memory bandwidth usage monitor capture instance selected by the Realm instance of [MSMON_CFG_MON_SEL](#).

If [MPAMF_IDR.HAS_RIS](#) is 1, the monitor instance long capture register accessed is for the resource instance currently selected by [MSMON_CFG_MON_SEL.RIS](#) and the monitor instance of that resource instance selected by [MSMON_CFG_MON_SEL.MON_SEL](#).

Configuration

The power domain of MSMON_MBWU_L_CAPTURE is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAM is implemented, MPAMF_IDR.HAS_MSMON == 1, MPAMF_MSMON_IDR.MSMON_MBWU == 1, MPAMF_MBWUMON_IDR.HAS_CAPTURE == 1 and MPAMF_MBWUMON_IDR.HAS_LONG == 1. Otherwise, direct accesses to MSMON_MBWU_L_CAPTURE are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU_L_CAPTURE is a 64-bit register.

Field descriptions

When MPAMF_MBWUMON_IDR.LWD == 0:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|--|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 | |
| NRDY | RES0 | | | | | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | |
| VALUE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |

NRDY, bit [63]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

| NRDY | Meaning |
|------|--|
| 0b0 | The captured monitor instance was ready and the MSMON_MBWU_L_CAPTURE.VALUE field is accurate. |
| 0b1 | The captured monitor instance was not ready and the contents of the MSMON_MBWU_L_CAPTURE.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage. |

Bits [62:44]

Reserved, RES0.

VALUE, bits [43:0]

Captured long memory bandwidth usage counter value if MSMON_MBWU_L_CAPTURE.NRDY is 0. Invalid if MSMON_MBWU_L_CAPTURE.NRDY is 1.

VALUE is the captured 44-bit count of bytes transferred since the monitor instance was last reset that met the criteria set in [MSMON_CFG_MBWU_FLT](#) and [MSMON_CFG_MBWU_CTL](#) for the monitor instance selected by [MSMON_CFG_MON_SEL](#).

When MPAMF_MBWUMON_IDR.LWD == 1:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| NRDY | VALUE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| VALUE | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

NRDY, bit [63]

Not Ready. Indicates whether the monitor has possibly inaccurate data.

| NRDY | Meaning |
|------|--|
| 0b0 | The captured monitor instance was ready and the MSMON_MBWU_L_CAPTURE.VALUE field is accurate. |
| 0b1 | The captured monitor instance was not ready and the contents of the MSMON_MBWU_L_CAPTURE.VALUE field might be inaccurate or otherwise not represent the actual memory bandwidth usage. |

VALUE, bits [62:0]

The captured long memory bandwidth usage counter value if MSMON_MBWU_L_CAPTURE.NRDY is 0. Invalid if MSMON_MBWU_L_CAPTURE.NRDY is 1.

VALUE is the captured 63-bit count of bytes transferred since the monitor instance was last reset that met the criteria set in [MSMON_CFG_MBWU_FLT](#) and [MSMON_CFG_MBWU_CTL](#) for the monitor instance selected by [MSMON_CFG_MON_SEL](#).

Accessing MSMON_MBWU_L_CAPTURE

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_MBWU_L_CAPTURE_s must be accessible from the Secure MPAM feature page.
- MSMON_MBWU_L_CAPTURE_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_MBWU_L_CAPTURE_rt must be accessible from the Root MPAM feature page.
- MSMON_MBWU_L_CAPTURE_rl must be accessible from the Realm MPAM feature page.

MSMON_MBWU_L_CAPTURE_s, MSMON_MBWU_L_CAPTURE_ns, MSMON_MBWU_L_CAPTURE_rt, and MSMON_MBWU_L_CAPTURE_rl must be separate registers:

- The Secure instance (MSMON_MBWU_L_CAPTURE_s) accesses the captured long memory bandwidth usage monitor used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_L_CAPTURE_ns) accesses the captured long memory bandwidth usage monitor used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_L_CAPTURE_rt) accesses the captured long memory bandwidth usage monitor used for Root PARTIDs.
- The Realm instance (MSMON_MBWU_L_CAPTURE_rl) accesses the captured long memory bandwidth usage monitor used for Realm PARTIDs.

When RIS is implemented, reads and writes to MSMON_MBWU_L_CAPTURE access the monitor instance for the bandwidth resource instance selected by [MSMON_CFG_MON_SEL](#).RIS and the memory bandwidth usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

When RIS is not implemented, reads and writes to MSMON_MBWU_L_CAPTURE access the monitor instance for the memory bandwidth usage monitor instance selected by [MSMON_CFG_MON_SEL](#).MON_SEL.

MSMON_MBWU_L_CAPTURE can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|----------------------|
| MPAM | MPAMF_BASE_s | 0x0890 | MSMON_MBWU_CAPTURE_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_ns | 0x0890 | MSMON_MBWU_CAPTURE_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rt | 0x0890 | MSMON_MBWU_CAPTURE_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-----------------------|
| MPAM | MPAMF_BASE_rl | 0x0890 | MSMON_MBWU_CAPTURE_rl |

When FEAT_RME is implemented access on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_MBWU_OFSR, MPAM MBWU Monitor Overflow Status Register

The MSMON_MBWU_OFSR characteristics are:

Purpose

MSMON_MBWU_OFSR is a 32-bit read-only register that shows bitmap of MBWU monitor instance overflow status for a contiguous group of 32 monitor instances.

MSMON_MBWU_OFSR_s gives a bitmap of pending MBWU overflow status for 32 Secure MBWU monitor instances. MSMON_MBWU_OFSR_ns gives a bitmap of pending MBWU overflow status for 32 Non-secure MBWU monitor instances. MSMON_MBWU_OFSR_rt gives a bitmap of pending MBWU overflow status for 32 Root MBWU monitor instances. MSMON_MBWU_OFSR_rl gives a bitmap of pending MBWU overflow status for 32 Realm MBWU monitor instances.

Configuration

The power domain of MSMON_MBWU_OFSR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_MBWUMON_IDR.HAS_OFSR == 1. Otherwise, direct accesses to MSMON_MBWU_OFSR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_MBWU_OFSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| OFPND31 | OFPND30 | OFPND29 | OFPND28 | OFPND27 | OFPND26 | OFPND25 | OFPND24 | OFPND23 | OFPND22 | OFPND21 | OFPND20 |

OFPND<i>, bit [i], for i = 31 to 0

Overflow status bitmap for MBWU monitor instances. The RIS and the contiguous range of MBWU monitor instances are set in [MSMON_CFG_MON_SEL](#). i of 0 corresponds to the MBWU monitor instance [MSMON_CFG_MON_SEL.MON_SEL & 0xFFE0](#).

| OFPND<i> | Meaning |
|----------|--|
| 0b0 | MBWU monitor instance (MSMON_CFG_MON_SEL.MON_SEL & 0xFFE0 + i) does not have a pending overflow. |
| 0b1 | MBWU monitor instance (MSMON_CFG_MON_SEL.MON_SEL & 0xFFE0 + i) has a pending overflow. |

After reading [MSMON_OFLOW_SR](#) to determine that an MBWU monitor instance has a pending overflow and which RIS values have pending overflows, an interrupt service routine could poll groups of 32 monitor instances in a RIS for pending monitors by reading this bitmap and incrementing [MSMON_CFG_MON_SEL.MON_SEL](#) by 32.

A pending overflow may be in either the [MSMON_CFG_MBWU_CTL.OFLOW_STATUS](#) or [MSMON_CFG_MBWU_CTL.OFLOW_STATUS_L](#) field.

Accessing MSMON_MBWU_OFSR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_MBWU_OFSR_s must be accessible from the Secure MPAM feature page.
- MSMON_MBWU_OFSR_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_MBWU_OFSR_rt must be accessible from the Root MPAM feature page.
- MSMON_MBWU_OFSR_rl must be accessible from the Realm MPAM feature page.

MSMON_MBWU_OFSR_s, MSMON_MBWU_OFSR_ns, MSMON_MBWU_OFSR_rt, and MSMON_MBWU_OFSR_rl must be separate registers:

- The Secure instance (MSMON_MBWU_OFSR_s) accesses the MBWU monitor overflow status bitmap used for Secure PARTIDs.
- The Non-secure instance (MSMON_MBWU_OFSR_ns) accesses the MBWU monitor overflow status bitmap used for Non-secure PARTIDs.
- The Root instance (MSMON_MBWU_OFSR_rt) accesses the MBWU monitor overflow status bitmap used for Root PARTIDs.
- The Realm instance (MSMON_MBWU_OFSR_rl) accesses the MBWU monitor overflow status bitmap used for Realm PARTIDs.

MSMON_MBWU_OFSR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|-------------------|
| MPAM | MPAMF_BASE_s | 0x0898 | MSMON_MBWU_OFSR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_ns | 0x0898 | MSMON_MBWU_OFSR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_rt | 0x0898 | MSMON_MBWU_OFSR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------|
| MPAM | MPAMF_BASE_rl | 0x0898 | MSMON_MBWU_OFSR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

MSMON_OFLOW_MSI_ADDR_H, MPAM Monitor Overflow MSI Write High-part Address Register

The MSMON_OFLOW_MSI_ADDR_H characteristics are:

Purpose

MSMON_OFLOW_MSI_ADDR_H is a 32-bit read/write register for the high part of the MPAM monitor overflow MSI address.

MSMON_OFLOW_MSI_ADDR_H_s is the high part of the MSI write address for monitor overflow interrupts from Secure monitor instances. MSMON_OFLOW_MSI_ADDR_H_ns is the high part of the MSI write address for monitor overflow interrupts from Non-secure monitor instances. MSMON_OFLOW_MSI_ADDR_H_rt is the high part of the MSI write address for monitor overflow interrupts from Root monitor instances. MSMON_OFLOW_MSI_ADDR_H_rl is the high part of the MSI write address for monitor overflow interrupts from Realm monitor instances.

Configuration

The power domain of MSMON_OFLOW_MSI_ADDR_H is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_ADDR_H are RES0.

[MSMON_OFLOW_MSI_ADDR_L](#), [MSMON_OFLOW_MSI_ADDR_H](#), [MSMON_OFLOW_MSI_ATTR](#), [MSMON_OFLOW_MSI_DATA](#), and [MSMON_OFLOW_MSI_MPAM](#) must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_ADDR_H is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | MSI_ADDR_H | | | | | | | | | | | | | | | | | | | |

Bits [31:20]

Reserved, RES0.

MSI_ADDR_H, bits [19:0]

MSI write address bits[51:32].

Accessing MSMON_OFLOW_MSI_ADDR_H

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_OFLOW_MSI_ADDR_H_s must be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_MSI_ADDR_H_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_MSI_ADDR_H_rt must be accessible from the Root MPAM feature page.
- MSMON_OFLOW_MSI_ADDR_H_rl must be accessible from the Realm MPAM feature page.

MSMON_OFLW_MSI_ADDR_H_s, MSMON_OFLW_MSI_ADDR_H_ns, MSMON_OFLW_MSI_ADDR_H_rt, and MSMON_OFLW_MSI_ADDR_H_rl must be separate registers:

- The Secure instance (MSMON_OFLW_MSI_ADDR_H_s) accesses the high part of the monitor overflow MSI write address of Secure monitors.
- The Non-secure instance (MSMON_OFLW_MSI_ADDR_H_ns) accesses the high part of the monitor overflow MSI write address of Non-secure monitors.
- The Root instance (MSMON_OFLW_MSI_ADDR_H_rt) accesses the high part of the monitor overflow MSI write address of Root monitors.
- The Realm instance (MSMON_OFLW_MSI_ADDR_H_rl) accesses the high part of the monitor overflow MSI write address of Realm monitors.

MSMON_OFLOW_MSI_ADDR_H can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|-------------------------|
| MPAM | MPAMF_BASE_s | 0x08E4 | MSMON_OFLW_MSI_ADDR_H_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------------|
| MPAM | MPAMF_BASE_ns | 0x08E4 | MSMON_OFLW_MSI_ADDR_H_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------------|
| MPAM | MPAMF_BASE_rt | 0x08E4 | MSMON_OFLW_MSI_ADDR_H_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|--------------------------|
| MPAM | MPAMF_BASE_rl | 0x08E4 | MSMON_OFLW_MSI_ADDR_H_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MSMON_OFLOW_MSI_ADDR_L, MPAM Monitor Overflow MSI Low-part Address Register

The MSMON_OFLOW_MSI_ADDR_L characteristics are:

Purpose

MSMON_OFLOW_MSI_ADDR_L is a 32-bit read/write register for the low part of the MPAM monitor MSI address.

MSMON_OFLOW_MSI_ADDR_L_s is the low part of the MSI write address for overflow interrupts from Secure monitor instances. MSMON_OFLOW_MSI_ADDR_L_ns is the low part of the MSI write address for overflow interrupts from Non-secure monitor instances. MSMON_OFLOW_MSI_ADDR_L_rt is the low part of the MSI write address for overflow interrupts from Root monitor instances. MSMON_OFLOW_MSI_ADDR_L_rl is the low part of the MSI write address for overflow interrupts from Realm monitor instances.

Configuration

The power domain of MSMON_OFLOW_MSI_ADDR_L is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_ADDR_L are RES0.

[MSMON_OFLOW_MSI_ADDR_L](#), [MSMON_OFLOW_MSI_ADDR_H](#), [MSMON_OFLOW_MSI_ATTR](#), [MSMON_OFLOW_MSI_DATA](#), and [MSMON_OFLOW_MSI_MPAM](#) must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_ADDR_L is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|-----------|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| MSI_ADDR_L | | | | | | | | | | | | | | | | | | | | | | | | | | | | | Bits[1:0] | | |

MSI_ADDR_L, bits [31:2]

MSI write address bits[31:2].

Bits [1:0]

Reads as 0b00.

Access to this field is **RO**.

Accessing MSMON_OFLOW_MSI_ADDR_L

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_OFLOW_MSI_ADDR_L_s must be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_MSI_ADDR_L_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_MSI_ADDR_L_rt must be accessible from the Root MPAM feature page.

- MSMON_OFLOW_MSI_ADDR_L_rl must be accessible from the Realm MPAM feature page.

MSMON_OFLOW_MSI_ADDR_L_s, MSMON_OFLOW_MSI_ADDR_L_ns, MSMON_OFLOW_MSI_ADDR_L_rt, and MSMON_OFLOW_MSI_ADDR_L_rl must be separate registers:

- The Secure instance (MSMON_OFLOW_MSI_ADDR_L_s) accesses the low part of the overflow MSI write address used for Secure PARTIDs.
- The Non-secure instance (MSMON_OFLOW_MSI_ADDR_L_ns) accesses the low part of the overflow MSI write address used for Non-secure PARTIDs.
- The Root instance (MSMON_OFLOW_MSI_ADDR_L_rt) accesses the low part of the overflow MSI write address used for Root PARTIDs.
- The Realm instance (MSMON_OFLOW_MSI_ADDR_L_rl) accesses the low part of the overflow MSI write address used for Realm PARTIDs.

MSMON_OFLOW_MSI_ADDR_L can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|--------------------------|
| MPAM | MPAMF_BASE_s | 0x08E0 | MSMON_OFLOW_MSI_ADDR_L_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------------|
| MPAM | MPAMF_BASE_ns | 0x08E0 | MSMON_OFLOW_MSI_ADDR_L_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------------|
| MPAM | MPAMF_BASE_rt | 0x08E0 | MSMON_OFLOW_MSI_ADDR_L_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|---------------------------|
| MPAM | MPAMF_BASE_rl | 0x08E0 | MSMON_OFLOW_MSI_ADDR_L_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MSMON_OFLOW_MSI_ATTR, MPAM Monitor Overflow MSI Write Attributes Register

The MSMON_OFLOW_MSI_ATTR characteristics are:

Purpose

MSMON_OFLOW_MSI_ATTR is a 32-bit read/write register that controls MPAM monitor overflow MSI write attributes for MPAM monitor overflows in this MSC.

MSMON_OFLOW_MSI_ATTR_s controls Secure MPAM monitor overflow MSI writes. MSMON_OFLOW_MSI_ATTR_ns controls Non-secure MPAM monitor overflow MSI writes. MSMON_OFLOW_MSI_ATTR_rt controls Root MPAM monitor overflow MSI writes. MSMON_OFLOW_MSI_ATTR_rl controls Realm MPAM monitor overflow MSI writes.

Configuration

The power domain of MSMON_OFLOW_MSI_ATTR is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_ATTR are RES0.

[MSMON_OFLOW_MSI_ADDR_L](#), [MSMON_OFLOW_MSI_ADDR_H](#), [MSMON_OFLOW_MSI_ATTR](#), [MSMON_OFLOW_MSI_DATA](#), and [MSMON_OFLOW_MSI_MPAM](#) must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_ATTR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|--------|-------------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | MSI_SH | MSI_MEMATTR | RES0 | | | | | | | | | | | | | | | | MSIEN | | | | | | | | | | | | |

Bits [31:30]

Reserved, RES0.

MSI_SH, bits [29:28]

Sharability attribute of MSI writes.

| MSI_SH | Meaning |
|--------|--------------------------------------|
| 0b00 | Non-shareable. |
| 0b01 | Reserved, CONSTRAINED UNPREDICTABLE. |
| 0b10 | Outer Shareable. |
| 0b11 | Inner Shareable. |

When MSMON_OFLOW_MSI_ATTR.MSI_MEMATTR specifies a Device memory type, the contents of this field are IGNORED and Shareability is effectively Outer Shareable.

MSI_MEMATTR, bits [27:24]

Memory attributes of MSI writes.

Note: This encoding matches the VMSAv8-64 stage 2 MemAttr[3:0] field as described in the Arm ARM, except that the following encodings are Reserved (not UNPREDICTABLE) and behave as Device-nGnRnE: 0b0100, 0b1000, and 0b1100.

| MSI_MEMATTR | Meaning |
|-------------|---|
| 0b0000 | Device-nGnRnE. |
| 0b0001 | Device-nGnRE. |
| 0b0010 | Device-nGRE. |
| 0b0011 | Device-GRE. |
| 0b0100 | Reserved. Behave as Device-nGnRnE, 0b0000. |
| 0b0101 | Normal Inner Non-cacheable, Outer Non-cacheable. |
| 0b0110 | Normal Inner Write-Through Cacheable, Outer Non-cacheable. |
| 0b0111 | Normal Inner Write-Back Cacheable, Outer Non-cacheable. |
| 0b1000 | Reserved. Behave as Device-nGnRnE, 0b0000. |
| 0b1001 | Normal Inner Non-Cachable, Outer Write-Through Cacheable. |
| 0b1010 | Normal Inner Write-Through Cacheable, Outer Write-Through Cachable. |
| 0b1011 | Normal Inner Write-Back Cacheable, Outer Write-Through Cachable. |
| 0b1100 | Reserved. Behave as Device-nGnRnE, 0b0000. |
| 0b1101 | Normal Inner Non-cacheable, Outer Write-Back Cacheable. |
| 0b1110 | Normal Inner Write-Through Cacheable, Outer Write-Back Cacheable. |
| 0b1111 | Normal Inner Write-Back Cacheable, Outer Write-Back Cacheable. |

When this field specifies a Device memory type, the contents of MSMON_OFLOW_MSI_ATTR.MSI_SH are IGNORED and Shareability is effectively Outer Shareable.

Device types may be implemented as any Device type with more n characters. For example, if this field is set to 0b0010, an implementation may treat the MSI write as the specified type, Device-nGRE, or as Device-nGnRE or as Device-nGnRnE.

Reserved encodings 0b0100, 0b1000, and 0b1100 must be implemented to behave the same as the 0b0000 encoding.

Bits [23:1]

Reserved, RES0.

MSIEN, bit [0]

Monitor overflow MSI write enable.

| MSIEN | Meaning |
|-------|---|
| 0b0 | MPAM monitor overflow MSI writes are not generated to signal enabled MPAM monitor overflow interrupts. When monitor overflow MSI writes are disabled, hardwired monitor overflow interrupt could be generated if hardwired monitor overflow interrupt is implemented. |
| 0b1 | MPAM monitor overflow MSI writes are generated to signal enabled MPAM monitor overflow interrupts. When monitor overflow MSI writes are enabled, hardwired monitor overflow interrupts are not generated. |

This enable affects whether a hardwired overflow interrupt is generated.

The reset behavior of this field is:

- On a MSC reset, this field resets to 0.

Accessing MSMON_OFLOW_MSI_ATTR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_OFLOW_MSI_ATTR_s must be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_MSI_ATTR_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_MSI_ATTR_rt must be accessible from the Root MPAM feature page.
- MSMON_OFLOW_MSI_ATTR_rl must be accessible from the Realm MPAM feature page.

MSMON_OFLOW_MSI_ATTR_s, MSMON_OFLOW_MSI_ATTR_ns, MSMON_OFLOW_MSI_ATTR_rt, and MSMON_OFLOW_MSI_ATTR_rl must be separate registers:

- The Secure instance (MSMON_OFLOW_MSI_ATTR_s) accesses the monitor overflow MSI write attributes of Secure monitors.
- The Non-secure instance (MSMON_OFLOW_MSI_ATTR_ns) accesses the monitor overflow MSI write attributes of Non-secure monitors.
- The Root instance (MSMON_OFLOW_MSI_ATTR_rt) accesses the monitor overflow MSI write attributes of Root monitors.
- The Realm instance (MSMON_OFLOW_MSI_ATTR_rl) accesses the monitor overflow MSI write attributes of Realm monitors.

MSMON_OFLOW_MSI_ATTR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|------------------------|
| MPAM | MPAMF_BASE_s | 0x08EC | MSMON_OFLOW_MSI_ATTR_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_ns | 0x08EC | MSMON_OFLOW_MSI_ATTR_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_rt | 0x08EC | MSMON_OFLOW_MSI_ATTR_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_rl | 0x08EC | MSMON_OFLOW_MSI_ATTR_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MSMON_OFLOW_MSI_DATA, MPAM Monitor Overflow MSI Write Data Register

The MSMON_OFLOW_MSI_DATA characteristics are:

Purpose

MSMON_OFLOW_MSI_DATA is a 32-bit read/write register for the MPAM monitor overflow MSI data.

MSMON_OFLOW_MSI_DATA_s is the data for the MSI write for monitor overflow from Secure monitor instances. MSMON_OFLOW_MSI_DATA_ns is the data for the MSI writes for monitor overflow interrupts from Non-secure monitor instances. MSMON_OFLOW_MSI_DATA_rt is the data for the MSI write for monitor overflow from Root monitor instances. MSMON_OFLOW_MSI_DATA_rl is the data for the MSI writes for monitor overflow interrupts from Realm monitor instances.

Configuration

The power domain of MSMON_OFLOW_MSI_DATA is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_DATA are RES0.

[MSMON_OFLOW_MSI_ADDR_L](#), [MSMON_OFLOW_MSI_ADDR_H](#), [MSMON_OFLOW_MSI_ATTR](#), [MSMON_OFLOW_MSI_DATA](#), and [MSMON_OFLOW_MSI_MPAM](#) must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_DATA is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| MSI_DATA | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

MSI_DATA, bits [31:0]

MSI write data word.

Accessing MSMON_OFLOW_MSI_DATA

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_OFLOW_MSI_DATA_s must be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_MSI_DATA_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_MSI_DATA_rt must be accessible from the Root MPAM feature page.
- MSMON_OFLOW_MSI_DATA_rl must be accessible from the Realm MPAM feature page.

MSMON_OFLOW_MSI_DATA_s, MSMON_OFLOW_MSI_DATA_ns, MSMON_OFLOW_MSI_DATA_rt, and MSMON_OFLOW_MSI_DATA_rl must be separate registers:

- The Secure instance (MSMON_OFLOW_MSI_DATA_s) accesses the monitor overflow MSI write data of Secure monitors.

- The Non-secure instance (MSMON_OFLOW_MSI_DATA_ns) accesses the monitor overflow MSI write data of Non-secure monitors.
- The Root instance (MSMON_OFLOW_MSI_DATA_rt) accesses the monitor overflow MSI write data of Root monitors.
- The Realm instance (MSMON_OFLOW_MSI_DATA_rl) accesses the monitor overflow MSI write data of Realm monitors.

MSMON_OFLOW_MSI_DATA can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|------------------------|
| MPAM | MPAMF_BASE_s | 0x08E8 | MSMON_OFLOW_MSI_DATA_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_ns | 0x08E8 | MSMON_OFLOW_MSI_DATA_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_rt | 0x08E8 | MSMON_OFLOW_MSI_DATA_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_rl | 0x08E8 | MSMON_OFLOW_MSI_DATA_rl |

When FEAT_RME is implemented access on this interface are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

MSMON_OFLOW_MSI_MPAM, MPAM Monitor Overflow MSI Write MPAM Information Register

The MSMON_OFLOW_MSI_MPAM characteristics are:

Purpose

MSMON_OFLOW_MSI_MPAM is a 32-bit read/write register that sets the MPAM information for a monitor overflow MSI write.

MSMON_OFLOW_MSI_MPAM_s controls MPAM information labeling of Secure monitor overflow MSI writes.
 MSMON_OFLOW_MSI_MPAM_ns controls MPAM information labeling of Non-secure monitor overflow MSI writes.
 MSMON_OFLOW_MSI_MPAM_rt controls MPAM information labeling of Root monitor overflow MSI writes.
 MSMON_OFLOW_MSI_MPAM_rl controls MPAM information labeling of Realm monitor overflow MSI writes.

Configuration

The power domain of MSMON_OFLOW_MSI_MPAM is IMPLEMENTATION DEFINED.

This register is present only when FEAT_MPAMv1p1 is implemented and MPAMF_MSMON_IDR.HAS_OFLW_MSI == 1. Otherwise, direct accesses to MSMON_OFLOW_MSI_MPAM are RES0.

[MSMON_OFLOW_MSI_ADDR_L](#), [MSMON_OFLOW_MSI_ADDR_H](#), [MSMON_OFLOW_MSI_ATTR](#), [MSMON_OFLOW_MSI_DATA](#), and [MSMON_OFLOW_MSI_MPAM](#) must all be implemented to support MSI writes for monitor overflow interrupts.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_MSI_MPAM is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|-----|----|----|----|----|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | PMG | | | | | | | | PARTID | | | | | | | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

PMG, bits [23:16]

Performance monitoring group property for an MSC monitor overflow MSI write.

The reset behavior of this field is:

- On a MSC reset, this field resets to an architecturally UNKNOWN value.

PARTID, bits [15:0]

Partition ID for an MSC monitor overflow MSI write.

The PARTID in this field is in the Secure PARTID space in the MSMON_OFLOW_MSI_MPAM_s instance and in the Non-secure PARTID space in the MSMON_OFLOW_MSI_MPAM_ns instance of this register.

The reset behavior of this field is:

- On a MSC reset, this field resets to an architecturally UNKNOWN value.

Accessing MSMON_OFLOW_MSI_MPAM

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- MSMON_OFLOW_MSI_MPAM_s must be accessible from the Secure MPAM feature page.
- MSMON_OFLOW_MSI_MPAM_ns must be accessible from the Non-secure MPAM feature page.
- MSMON_OFLOW_MSI_MPAM_rt must be accessible from the Root MPAM feature page.
- MSMON_OFLOW_MSI_MPAM_rl must be accessible from the Realm MPAM feature page.

MSMON_OFLOW_MSI_MPAM_s, MSMON_OFLOW_MSI_MPAM_ns, MSMON_OFLOW_MSI_MPAM_rt, and MSMON_OFLOW_MSI_MPAM_rl must be separate registers:

- The Secure instance (MSMON_OFLOW_MSI_MPAM_s) accesses the monitor overflow MSI MPAM information of Secure monitors.
- The Non-secure instance (MSMON_OFLOW_MSI_MPAM_ns) accesses the monitor overflow MSI MPAM information of Non-secure monitors.
- The Root instance (MSMON_OFLOW_MSI_MPAM_rt) accesses the monitor overflow MSI MPAM information of Root monitors.
- The Realm instance (MSMON_OFLOW_MSI_MPAM_rl) accesses the monitor overflow MSI MPAM information of Realm monitors.

MSMON_OFLOW_MSI_MPAM can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|------------------------|
| MPAM | MPAMF_BASE_s | 0x08DC | MSMON_OFLOW_MSI_MPAM_s |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_ns | 0x08DC | MSMON_OFLOW_MSI_MPAM_ns |

Accesses on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_rt | 0x08DC | MSMON_OFLOW_MSI_MPAM_rt |

When FEAT_RME is implemented access on this interface are **RW**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------------|
| MPAM | MPAMF_BASE_rl | 0x08DC | MSMON_OFLOW_MSI_MPAM_rl |

When FEAT_RME is implemented access on this interface are **RW**.

MSMON_OFLOW_SR, MPAM Monitor Overflow Status Register

The MSMON_OFLOW_SR characteristics are:

Purpose

MSMON_OFLOW_SR is a 32-bit read-only register that shows MPAM monitor overflow status for this MSC.

MSMON_OFLOW_SR_s gives the status of overflows of Secure MPAM monitors. MSMON_OFLOW_SR_ns gives the status of overflows of Non-secure MPAM monitors. MSMON_OFLOW_SR_rt gives the status of overflows of Root MPAM monitors. MSMON_OFLOW_SR_rl gives the status of overflows of Realm MPAM monitors.

Configuration

The power domain of MSMON_OFLOW_SR is IMPLEMENTATION DEFINED.

This register is present only when MPAMF_MSMON_IDR.HAS_OFLOW_SR == 1. Otherwise, direct accesses to MSMON_OFLOW_SR are RES0.

The power and reset domain of each MSC component is specific to that component.

Attributes

MSMON_OFLOW_SR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------------|----------------|------|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|-----------|-----------|-----------|-----------|-----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CSU_OFLOW_PND | MBWU_OFLOW_PND | RES0 | | | | | | | | | | | | | | RIS_PND15 | RIS_PND14 | RIS_PND13 | RIS_PND12 | RIS_PND11 | RIS_PND10 | RIS_PND9 | RIS_PND8 | RIS_PND7 | RIS_PND6 | RIS_PND5 | RIS_PND4 | RIS_PND3 | RIS_PND2 | RIS_PND1 | RIS_PND0 |

CSU_OFLOW_PND, bit [31]

At least one cache storage usage monitor has OFLOW_STATUS of 1 in [MSMON_CFG_CSU_CTL](#).

| CSU_OFLOW_PND | Meaning |
|---------------|--|
| 0b0 | There are no cache storage usage monitor instances where MSMON_CFG_CSU_CTL .OFLOW_STATUS is 1. |
| 0b1 | MSMON_CFG_CSU_CTL for at least one of the cache storage usage monitor instances has OFLOW_STATUS set to 1. |

This field clears when [MSMON_CFG_CSU_CTL](#).OFLOW_STATUS has been reset to 0 for all CSU monitor instances in this MSC.

MBWU_OFLOW_PND, bit [30]

At least one memory bandwidth usage monitor instance has OFLOW_STATUS or OFLOW_STATUS_L of 1 in [MSMON_CFG_MBWU_CTL](#).

| MBWU_OFLOW_PND | Meaning |
|----------------|---|
| 0b0 | There are no memory bandwidth usage monitor instances where MSMON_CFG_MBWU_CTL.OFLOW_STATUS is 1. |
| 0b1 | MSMON_CFG_MBWU_CTL for at least one of the memory bandwidth usage monitor instances has either OFLOW_STATUS or OFLOW_STATUS_L set to 1. |

This field clears when [MSMON_CFG_MBWU_CTL.OFLOW_STATUS](#) and [MSMON_CFG_MBWU_CTL.OFLOW_STATUS_L](#) have been reset to 0 for all MBWU monitor instances in this MSC.

Bits [29:16]

Reserved, RES0.

RIS_PND<r>, bit [r], for r = 15 to 0

Overflow status by RIS.

| RIS_PND<r> | Meaning |
|------------|--|
| 0b0 | RIS r has no unread overflows of any type of monitor. |
| 0b1 | RIS r has at least one unread overflow in at least one of the monitor types. |

Combined with the CSU_OFLOW_PND and MBWU_OFLOW_PND flags in this register, an interrupt service routine could poll only the monitor types indicated in monitors for the resource instances flagged in this field.

Bit r is set when any monitor instance of any type in resource instance r has OFLOW_STATUS or OFLOW_STATUS_L set to 1.

Accessing MSMON_OFLOW_SR

This register is within the MPAM feature page memory frames.

In a system that supports Secure, Non-secure, Root, and Realm memory maps, there must be MPAM feature pages in all four address maps:

- [MSMON_OFLOW_SR_s](#) must be accessible from the Secure MPAM feature page.
- [MSMON_OFLOW_SR_ns](#) must be accessible from the Non-secure MPAM feature page.
- [MSMON_OFLOW_SR_rt](#) must be accessible from the Root MPAM feature page.
- [MSMON_OFLOW_SR_rl](#) must be accessible from the Realm MPAM feature page.

[MSMON_OFLOW_SR_s](#), [MSMON_OFLOW_SR_ns](#), [MSMON_OFLOW_SR_rt](#), and [MSMON_OFLOW_SR_rl](#) must be separate registers:

- The Secure instance ([MSMON_OFLOW_SR_s](#)) accesses the monitor overflow status summary of Secure monitors.
- The Non-secure instance ([MSMON_OFLOW_SR_ns](#)) accesses the monitor overflow status summary of Non-secure monitors.
- The Root instance ([MSMON_OFLOW_SR_rt](#)) accesses the monitor overflow status summary of Root monitors.
- The Realm instance ([MSMON_OFLOW_SR_rl](#)) accesses the monitor overflow status summary of Realm monitors.

MSMON_OFLOW_SR can be accessed through the memory-mapped interfaces:

| Component | Frame | Offset | Instance |
|-----------|--------------|--------|------------------|
| MPAM | MPAMF_BASE_s | 0x08F0 | MSMON_OFLOW_SR_s |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_ns | 0x08F0 | MSMON_OFLOW_SR_ns |

Accesses on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_rt | 0x08F0 | MSMON_OFLOW_SR_rt |

When FEAT_RME is implemented access on this interface are **RO**.

| Component | Frame | Offset | Instance |
|-----------|---------------|--------|-------------------|
| MPAM | MPAMF_BASE_rl | 0x08F0 | MSMON_OFLOW_SR_rl |

When FEAT_RME is implemented access on this interface are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

OSLAR_EL1, OS Lock Access Register

The OSLAR_EL1 characteristics are:

Purpose

Used to lock or unlock the OS Lock.

Configuration

External register OSLAR_EL1 bits [31:0] are architecturally mapped to AArch64 System register [OSLAR_EL1\[31:0\]](#).

OSLAR_EL1 is in the Core power domain.

The OS Lock can also be locked or unlocked using [DBGOSLAR](#).

If FEAT_Debugv8p2 is not implemented, it is IMPLEMENTATION DEFINED whether external debug accesses to OSLAR_EL1 are ignored and return an error when AllowExternalDebugAccess() returns FALSE for the access.

If FEAT_Debugv8p2 is implemented, external debug accesses to OSLAR_EL1 are ignored and return an error when AllowExternalDebugAccess() returns FALSE for the access.

Attributes

OSLAR_EL1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | OSLK | | | | | | | | | | | | | | | |

Bits [31:1]

Reserved, RES0.

OSLK, bit [0]

On writes to OSLAR_EL1, bit[0] is copied to the OS Lock.

Use [EDPRSR.OSLK](#) to check the current status of the lock.

Accessing OSLAR_EL1

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalDebugAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

OSLAR_EL1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| Debug | 0x300 | OSLAR_EL1 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), AllowExternalDebugAccess() and SoftwareLockStatus() accesses to this register are **WI**.
- When IsCorePowered(), !DoubleLockStatus(), AllowExternalDebugAccess() and !SoftwareLockStatus() accesses to this register are **WO**.
- When IsCorePowered(), !DoubleLockStatus(), !AllowExternalDebugAccess() and FEAT_Debugv8p2 is not implemented accesses to this register are **IMPDEF**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMAUTHSTATUS, Performance Monitors Authentication Status register

The PMAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for Performance Monitors.

Configuration

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is OPTIONAL, and is required for CoreSight compliance. Arm recommends that this register is implemented.

Attributes

PMAUTHSTATUS is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|-------|------|------|----|----|----|-------|------|------|----|----|----|------|-----|-------|------|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | RTNID | RTID | RES0 | | | | RLNID | RLID | RES0 | | | | SNID | SID | NSNID | NSID | | | | | | | | | | | | |

Bits [31:28]

Reserved, RES0.

RTNID, bits [27:26]

Root non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RTNID.

RTID, bits [25:24]

Root invasive debug.

| RTID | Meaning |
|------|------------------|
| 0b00 | Not implemented. |

Bits [23:16]

Reserved, RES0.

RLNID, bits [15:14]

Realm non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RLNID.

RLID, bits [13:12]

Realm invasive debug.

| RLID | Meaning |
|------|------------------|
| 0b00 | Not implemented. |

Bits [11:8]

Reserved, RES0.

SNID, bits [7:6]

Holds the same value as [DBGAUTHSTATUS_EL1](#).SNID.

SID, bits [5:4]

Secure invasive debug. Possible values of this field are:

| SID | Meaning |
|------|------------------|
| 0b00 | Not implemented. |

All other values are reserved.

NSNID, bits [3:2]

Holds the same value as [DBGAUTHSTATUS_EL1](#).NSNID.

NSID, bits [1:0]

Non-secure invasive debug. Possible values of this field are:

| NSID | Meaning |
|------|------------------|
| 0b00 | Not implemented. |

All other values are reserved.

Accessing PMAUTHSTATUS

PMAUTHSTATUS can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|--------------|
| PMU | 0xFB8 | PMAUTHSTATUS |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

PMCCFILTR_EL0, Performance Monitors Cycle Counter Filter Register

The PMCCFILTR_EL0 characteristics are:

Purpose

Determines the modes in which the Cycle Counter, [PMCCNTR_EL0](#), increments.

Configuration

External register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to AArch64 System register [PMCCFILTR_EL0\[31:0\]](#).

External register PMCCFILTR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMCCFILTR\[31:0\]](#).

PMCCFILTR_EL0 is in the Core power domain.

On a Warm or Cold reset, RW fields in this register reset to:

- Architecturally UNKNOWN values if the reset is to an Exception level that is using AArch64.
- 0 if the reset is to an Exception level that is using AArch32.

The register is not affected by an External debug reset.

Attributes

PMCCFILTR_EL0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|-----|-----|-----|----|------|----|------|-----|-----|-----|------|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P | U | NSK | NSU | NSH | M | RES0 | SH | RES0 | RLK | RLU | RLH | RES0 | | | | | | | | | | | | | | | | | | | |

P, bit [31]

Privileged filtering bit. Controls counting in EL1.

If EL3 is implemented, then counting in Non-secure EL1 is further controlled by the PMCCFILTR_EL0.NSK bit.

If FEAT_RME is implemented, then counting in Realm EL1 is further controlled by the PMCCFILTR_EL0.RLK bit.

| P | Meaning |
|-----|-----------------------------|
| 0b0 | Count cycles in EL1. |
| 0b1 | Do not count cycles in EL1. |

U, bit [30]

User filtering bit. Controls counting in EL0.

If EL3 is implemented, then counting in Non-secure EL0 is further controlled by the PMCCFILTR_EL0.NSU bit.

If FEAT_RME is implemented, then counting in Realm EL0 is further controlled by the PMCCFILTR_EL0.RLU bit.

| U | Meaning |
|-----|-----------------------------|
| 0b0 | Count cycles in EL0. |
| 0b1 | Do not count cycles in EL0. |

NSK, bit [29]**When EL3 is implemented:**

Non-secure EL1 (kernel) modes filtering bit. Controls counting in Non-secure EL1.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.P bit, cycles in Non-secure EL1 are counted.

Otherwise, cycles in Non-secure EL1 are not counted.

Otherwise:

Reserved, RES0.

NSU, bit [28]**When EL3 is implemented:**

Non-secure EL0 (Unprivileged) filtering bit. Controls counting in Non-secure EL0.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.U bit, cycles in Non-secure EL0 are counted.

Otherwise, cycles in Non-secure EL0 are not counted.

Otherwise:

Reserved, RES0.

NSH, bit [27]**When EL2 is implemented:**

EL2 (Hypervisor) filtering bit. Controls counting in EL2.

If FEAT_SEL2 and EL3 are implemented, counting in Secure EL2 is further controlled by the PMCCFILTR_EL0.SH bit.

If FEAT_RME is implemented, then counting in Realm EL2 is further controlled by the PMCCFILTR_EL0.RLH bit.

| NSH | Meaning |
|-----|-----------------------------|
| 0b0 | Do not count cycles in EL2. |
| 0b1 | Count cycles in EL2. |

Otherwise:

Reserved, RES0.

M, bit [26]**When EL3 is implemented:**

Secure EL3 filtering bit.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.P bit, cycles in Secure EL3 are counted.

Otherwise, cycles in Secure EL3 are not counted.

Most applications can ignore this field and set its value to 0.

Note

This field is not visible in the AArch32 [PMCCFILTR](#) System register.

Otherwise:

Reserved, RES0.

Bit [25]

Reserved, RES0.

SH, bit [24]

When FEAT_SEL2 is implemented and EL3 is implemented:

Secure EL2 filtering.

If the value of this bit is not equal to the value of the PMCCFILTR_EL0.NSH bit, cycles in Secure EL2 are counted.

Otherwise, cycles in Secure EL2 are not counted.

If Secure EL2 is disabled, this field is RES0.

Note

This field is not visible in the AArch32 [PMCCFILTR](#) System register.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES0.

RLK, bit [22]

When FEAT_RME is implemented:

Realm EL1 (kernel) filtering bit. Controls counting in Realm EL1.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.P bit, cycles in Realm EL1 are counted.

Otherwise, cycles in Realm EL1 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLU, bit [21]

When FEAT_RME is implemented:

Realm EL0 (unprivileged) filtering bit. Controls counting in Realm EL0.

If the value of this bit is equal to the value of the PMCCFILTR_EL0.U bit, cycles in Realm EL0 are counted.

Otherwise, cycles in Realm EL0 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLH, bit [20]**When FEAT_RME is implemented:**

Realm EL2 filtering bit. Controls counting in Realm EL2.

If the value of this bit is not equal to the value of the PMCCFILTR_EL0.NSH bit, cycles in Realm EL2 are counted.

Otherwise, cycles in Realm EL2 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:0]

Reserved, RES0.

Accessing PMCCFILTR_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCCFILTR_EL0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|---------------|
| PMU | 0x47C | PMCCFILTR_EL0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

PMCCNTR_EL0, Performance Monitors Cycle Counter

The PMCCNTR_EL0 characteristics are:

Purpose

Holds the value of the processor Cycle Counter, CCNT, that counts processor clock cycles. For more information, see 'Time as measured by the Performance Monitors cycle counter'.

[PMCCFILTR_EL0](#) determines the modes and states in which the PMCCNTR_EL0 can increment.

Configuration

External register PMCCNTR_EL0 bits [63:0] are architecturally mapped to AArch64 System register [PMCCNTR_EL0\[63:0\]](#).

External register PMCCNTR_EL0 bits [63:0] are architecturally mapped to AArch32 System register [PMCCNTR\[63:0\]](#).

PMCCNTR_EL0 is in the Core power domain.

Attributes

PMCCNTR_EL0 is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | CCNT | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | CCNT | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

CCNT, bits [63:0]

Cycle count. Depending on the values of [PMCR_EL0](#).{LC,D}, the cycle count increments in one of the following ways:

- Every processor clock cycle.
- Every 64th processor clock cycle.

Writing 1 to [PMCR_EL0](#).C sets this field to 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCCNTR_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCCNTR_EL0 can be accessed through the external debug interface:

| Component | Offset | Instance | Range |
|-----------|--------|----------|-------|
|-----------|--------|----------|-------|

| | | | |
|-----|-------|-------------|------|
| PMU | 0x0F8 | PMCCNTR_EL0 | 31:0 |
|-----|-------|-------------|------|

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

| Component | Offset | Instance | Range |
|-----------|--------|-------------|-------|
| PMU | 0x0FC | PMCCNTR_EL0 | 63:32 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID0, Performance Monitors Common Event Identification register 0

The PMCEID0 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range 0x0000 to 0x001F.

For more information about the Common events and the use of the PMCEIDn registers, see 'The PMU event number space and common events'.

Note

This view of the register was previously called PMCEID0_EL0.

Configuration

External register PMCEID0 bits [31:0] are architecturally mapped to AArch64 System register [PMCEID0_EL0\[31:0\]](#).

External register PMCEID0 bits [31:0] are architecturally mapped to AArch32 System register [PMCEID0\[31:0\]](#).

PMCEID0 is in the Core power domain.

Attributes

PMCEID0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|---------------------|---------------------|---------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 |
| ID31 | ID30 | ID29 | ID28 | ID27 | ID26 | ID25 | ID24 | ID23 | ID22 | ID21 | ID20 | ID19 | ID18 | ID17 | ID16 | ID15 | ID14 | ID13 | ID12 | ID11 | ID10 | ID9 | ID8 | ID7 |

ID<n>, bit [n], for n = 31 to 0

ID[n] corresponds to Common event n.

For each bit:

| ID<n> | Meaning |
|-------|--|
| 0b0 | The Common event is not implemented, or not counted. |
| 0b1 | The Common event is implemented. |

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID0

Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCEID0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xE20 | PMCEID0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and AllowExternalPMUAccess() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID1, Performance Monitors Common Event Identification register 1

The PMCEID1 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range 0x020 to 0x03F.

For more information about the Common events and the use of the PMCEIDn registers, see 'The PMU event number space and common events'.

Note

This view of the register was previously called PMCEID1_EL0.

Configuration

External register PMCEID1 bits [31:0] are architecturally mapped to AArch64 System register [PMCEID1_EL0\[31:0\]](#).

External register PMCEID1 bits [31:0] are architecturally mapped to AArch32 System register [PMCEID1\[31:0\]](#).

PMCEID1 is in the Core power domain.

Attributes

PMCEID1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|----------------------|---------------------|---------------------|---------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 |
| ID31 | ID30 | ID29 | ID28 | ID27 | ID26 | ID25 | ID24 | ID23 | ID22 | ID21 | ID20 | ID19 | ID18 | ID17 | ID16 | ID15 | ID14 | ID13 | ID12 | ID11 | ID10 | ID9 | ID8 | ID7 |

ID<n>, bit [n], for n = 31 to 0

ID[n] corresponds to Common event (0x0020 + n).

For each bit:

| ID<n> | Meaning |
|-------|--|
| 0b0 | The Common event is not implemented, or not counted. |
| 0b1 | The Common event is implemented. |

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID1

Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCEID1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xE24 | PMCEID1 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and AllowExternalPMUAccess() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID2, Performance Monitors Common Event Identification register 2

The PMCEID2 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range 0x4000 to 0x401F.

For more information about the Common events and the use of the PMCEIDn registers, see 'The PMU event number space and common events'.

Configuration

External register PMCEID2 bits [31:0] are architecturally mapped to AArch64 System register [PMCEID0_EL0\[63:32\]](#).

External register PMCEID2 bits [31:0] are architecturally mapped to AArch32 System register [PMCEID2\[31:0\]](#).

PMCEID2 is in the Core power domain.

This register is present only when FEAT_PMUv3p1 is implemented. Otherwise, direct accesses to PMCEID2 are RES0.

Attributes

PMCEID2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | |
|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 |
| IDHi31 | IDHi30 | IDHi29 | IDHi28 | IDHi27 | IDHi26 | IDHi25 | IDHi24 | IDHi23 | IDHi22 | IDHi21 | IDHi20 | IDHi19 | IDHi18 | IDHi17 | IDHi16 | IDHi15 |

IDHi<n>, bit [n], for n = 31 to 0

IDHi[n] corresponds to Common event (0x4000 + n).

For each bit:

| IDHi<n> | Meaning |
|---------|--|
| 0b0 | The Common event is not implemented, or not counted. |
| 0b1 | The Common event is implemented. |

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU

features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID2

Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCEID2 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xE28 | PMCEID2 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and AllowExternalPMUAccess() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCEID3, Performance Monitors Common Event Identification register 3

The PMCEID3 characteristics are:

Purpose

Defines which Common architectural events and Common microarchitectural events are implemented, or counted, using PMU events in the range 0x4020 to 0x403F.

For more information about the Common events and the use of the PMCEIDn registers, see 'The PMU event number space and common events'.

Configuration

External register PMCEID3 bits [31:0] are architecturally mapped to AArch64 System register [PMCEID1_EL0\[63:32\]](#).

External register PMCEID3 bits [31:0] are architecturally mapped to AArch32 System register [PMCEID3\[31:0\]](#).

PMCEID3 is in the Core power domain.

This register is present only when FEAT_PMUv3p1 is implemented. Otherwise, direct accesses to PMCEID3 are RES0.

Attributes

PMCEID3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | |
|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 |
| IDhi31 | IDhi30 | IDhi29 | IDhi28 | IDhi27 | IDhi26 | IDhi25 | IDhi24 | IDhi23 | IDhi22 | IDhi21 | IDhi20 | IDhi19 | IDhi18 | IDhi17 | IDhi16 | IDhi15 |

IDhi<n>, bit [n], for n = 31 to 0

IDhi[n] corresponds to Common event (0x4020 + n).

For each bit:

| IDhi<n> | Meaning |
|---------|--|
| 0b0 | The Common event is not implemented, or not counted. |
| 0b1 | The Common event is implemented. |

When the value of a bit in the field is 1, the corresponding Common event is implemented and counted.

Note

Arm recommends that if a Common event is never counted, the value of the corresponding bit is 0.

A bit that corresponds to a reserved event number is reserved. The value might be used in a future revision of the architecture to identify an additional Common event.

Note

Such an event might be added retrospectively to an earlier version of the PMU architecture, provided the event does not require any additional PMU

features and has an event number that can be represented in the PMCEID<n> registers of that earlier version of the PMU architecture.

Accessing PMCEID3

Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCEID3 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xE2C | PMCEID3 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and AllowExternalPMUAccess() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCFGR, Performance Monitors Configuration Register

The PMCFGR characteristics are:

Purpose

Contains PMU-specific configuration data.

Configuration

PMCFGR is in the Core power domain.

Attributes

PMCFGR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----|----|----|----|------|----|----|----|-----|----|----|----|------|-----|------|----|-----|----|------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NCG | | | | RES0 | | | | FZO | | | | RES0 | UEN | WTNA | EX | CCD | CC | SIZE | | | | N | | | | | | | | | |

NCG, bits [31:28]

This feature is not supported, so this field is RAZ.

Reads as 0b0000.

Access to this field is **RO**.

Bits [27:22]

Reserved, RES0.

FZO, bit [21]

Freeze-on-overflow supported. Defined values are:

| FZO | Meaning |
|-----|--|
| 0b0 | Freeze-on-overflow mechanism is not supported. PMCR_ELO .FZO is RES0. |
| 0b1 | Freeze-on-overflow mechanism is supported. PMCR_ELO .FZO is RW. |

FEAT_PMUv3p7 implements the functionality added by the value 0b1.

From Armv8.7, if FEAT_PMUv3 is implemented, the only permitted value is 0b1.

Bit [20]

Reserved, RES0.

UEN, bit [19]

User-mode Enable Register supported. [PMUSERENR_ELO](#) is not visible in the external debug interface, so this bit is RAZ.

Reads as 0b0.

Access to this field is **RO**.

WT, bit [18]

This feature is not supported, so this bit is RAZ.

Reads as 0b0.

Access to this field is **RO**.

NA, bit [17]

This feature is not supported, so this bit is RAZ.

Reads as 0b0.

Access to this field is **RO**.

EX, bit [16]

Export supported. Value is IMPLEMENTATION DEFINED.

| EX | Meaning |
|-----|--|
| 0b0 | PMCR_ELO .X is RES0. |
| 0b1 | PMCR_ELO .X is read/write. |

Access to this field is **RO**.

CCD, bit [15]

Cycle counter has prescale.

This is RES1 if AArch32 is supported, and RAZ otherwise.

| CCD | Meaning |
|-----|--|
| 0b0 | PMCR_ELO .D is RES0. |
| 0b1 | PMCR_ELO .D is read/write. |

CC, bit [14]

Dedicated cycle counter (counter 31) supported.

Reads as 0b1.

Access to this field is **RO**.

SIZE, bits [13:8]

Size of counters, minus one. This field defines the size of the largest counter implemented by the Performance Monitors Unit.

From Armv8, the largest counter is 64-bits, so the value of this field is 0b111111.

This field is used by software to determine the spacing of the counters in the memory-map. From Armv8, the counters are a doubleword-aligned addresses.

Reads as 0b111111.

Access to this field is **RO**.

N, bits [7:0]

Number of counters implemented in addition to the cycle counter, [PMCCNTR_EL0](#). The maximum number of event counters is 31.

| N | Meaning |
|------|---|
| 0x00 | Only PMCCNTR_EL0 implemented. |
| 0x01 | PMCCNTR_EL0 plus one event counter implemented. |

and so on up to 0b00011111, which indicates [PMCCNTR_EL0](#) and 31 event counters implemented.

Accessing PMCFGR

Note

AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCFGR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xE00 | PMCFGR |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and AllowExternalPMUAccess() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCID1SR, CONTEXTIDR_EL1 Sample Register

The PMCID1SR characteristics are:

Purpose

Contains the sampled value of [CONTEXTIDR_EL1](#), captured on reading [PMPCSR](#)[31:0].

Configuration

PMCID1SR is in the Core power domain.

This register is present only when FEAT_PCSRv8p2 is implemented. Otherwise, direct accesses to PMCID1SR are RES0.

Note

Before Armv8.2, the PC Sample-based Profiling Extension can be implemented in the external debug register space, as indicated by the value of [EDDEVID](#).PCSample.

Attributes

PMCID1SR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CONTEXTIDR_EL1 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

CONTEXTIDR_EL1, bits [31:0]

Context ID. The value of CONTEXTIDR that is associated with the most recent [PMPCSR](#) sample. When the most recent [PMPCSR](#) sample was generated:

- If EL1 is using AArch64, then the Context ID is sampled from [CONTEXTIDR_EL1](#).
- If EL1 is using AArch32, then the Context ID is sampled from [CONTEXTIDR](#).
- If EL3 is implemented and is using AArch32, then [CONTEXTIDR](#) is a banked register and PMCID1SR samples the current banked copy of [CONTEXTIDR](#) for the Security state that is associated with the most recent [PMPCSR](#) sample.

Because the value written to PMCID1SR is an indirect read of CONTEXTIDR, it is CONSTRAINED UNPREDICTABLE whether PMCID1SR is set to the original or new value if [PMPCSR](#) samples:

- An instruction that writes to CONTEXTIDR.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCID1SR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

PMCID1SR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0x208 | PMCID1SR |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0x228 | PMCID1SR |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCID2SR, CONTEXTIDR_EL2 Sample Register

The PMCID2SR characteristics are:

Purpose

Contains the sampled value of [CONTEXTIDR_EL2](#), captured on reading [PMPCSR](#)[31:0].

Configuration

PMCID2SR is in the Core power domain.

This register is present only when FEAT_PCSRv8p2 is implemented and EL2 is implemented. Otherwise, direct accesses to PMCID2SR are RES0.

Note

If FEAT_PCSRv8p2 is not implemented, the PC Sample-based Profiling Extension can be implemented in the external debug register space, as indicated by the value of [EDDEVID](#).PCSample.

Attributes

PMCID2SR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| CONTEXTIDR_EL2 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

CONTEXTIDR_EL2, bits [31:0]

Context ID. The value of [CONTEXTIDR_EL2](#) that is associated with the most recent [PMPCSR](#) sample. When the most recent [PMPCSR](#) sample is generated:

- If the PE is not executing at EL3, EL2 is using AArch64, and EL2 is enabled in the current Security state, then this field is set to the Context ID sampled from [CONTEXTIDR_EL2](#).
- Otherwise, this field is set to an UNKNOWN value.

Because the value written to PMCID2SR is an indirect read of [CONTEXTIDR_EL2](#), it is CONSTRAINED UNPREDICTABLE whether PMCID2SR is set to the original or new value if [PMPCSR](#) samples:

- An instruction that writes to [CONTEXTIDR_EL2](#).
- The next Context synchronization event.
- Any instruction executed between these two instructions.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCID2SR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

PMCID2SR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0x22C | PMCID2SR |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCIDR0, Performance Monitors Component Identification Register 0

The PMCIDR0 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Component Identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

PMCIDR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL_0 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Preamble.

Reads as 0x0D.

Access to this field is **RO**.

Accessing PMCIDR0

PMCIDR0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xFF0 | PMCIDR0 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

PMCIDR1, Performance Monitors Component Identification Register 1

The PMCIDR1 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Component Identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

PMCIDR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | CLASS | | | PRMBL_1 | | | | |

Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

| CLASS | Meaning |
|--------|----------------------|
| 0b1001 | CoreSight component. |

Other values are defined by the CoreSight Architecture.

This field reads as 0x9.

PRMBL_1, bits [3:0]

Preamble. RAZ.

Reads as 0b0000.

Access to this field is **RO**.

Accessing PMCIDR1

PMCIDR1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xFF4 | PMCIDR1 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCIDR2, Performance Monitors Component Identification Register 2

The PMCIDR2 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Component Identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

PMCIDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL_2 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Preamble.

Reads as 0x05.

Access to this field is **RO**.

Accessing PMCIDR2

PMCIDR2 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xFF8 | PMCIDR2 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

PMCIDR3, Performance Monitors Component Identification Register 3

The PMCIDR3 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Component Identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

PMCIDR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL_3 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Preamble.

Reads as 0xB1.

Access to this field is **RO**.

Accessing PMCIDR3

PMCIDR3 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xFFC | PMCIDR3 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

PMCNTENCLR_EL0, Performance Monitors Count Enable Clear register

The PMCNTENCLR_EL0 characteristics are:

Purpose

Disables the Cycle Count Register, [PMCCNTR_EL0](#), and any implemented event counters [PMEVCNTR<n>](#). Reading this register shows which counters are enabled.

Configuration

External register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch64 System register [PMCNTENCLR_EL0\[31:0\]](#).

External register PMCNTENCLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMCNTENCLR\[31:0\]](#).

PMCNTENCLR_EL0 is in the Core power domain.

Attributes

PMCNTENCLR_EL0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

C, bit [31]

[PMCCNTR_EL0](#) disable bit. Disables the cycle counter register. Possible values are:

| C | Meaning |
|-----|--|
| 0b0 | When read, means the cycle counter is disabled. When written, has no effect. |
| 0b1 | When read, means the cycle counter is enabled. When written, disables the cycle counter. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter disable bit for [PMEVCNTR<n>_EL0](#).

If [PMCFGR.N](#) is less than 31, bits [30:[PMCFGR.N](#)] are RAZ/WI.

| P<n> | Meaning |
|------|--|
| 0b0 | When read, means that PMEVCNTR<n>_EL0 is disabled. When written, has no effect. |
| 0b1 | When read, means that PMEVCNTR<n>_EL0 is enabled. When written, disables PMEVCNTR<n>_EL0 . |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCNTENCLR_ELO

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCNTENCLR_ELO can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------------|
| PMU | 0xC20 | PMCNTENCLR_ELO |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCNTENSET_EL0, Performance Monitors Count Enable Set register

The PMCNTENSET_EL0 characteristics are:

Purpose

Enables the Cycle Count Register, [PMCCNTR_EL0](#), and any implemented event counters [PMEVCNTR<n>](#). Reading this register shows which counters are enabled.

Configuration

External register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch64 System register [PMCNTENSET_EL0\[31:0\]](#).

External register PMCNTENSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMCNTENSET\[31:0\]](#).

PMCNTENSET_EL0 is in the Core power domain.

Attributes

PMCNTENSET_EL0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

C, bit [31]

[PMCCNTR_EL0](#) enable bit. Enables the cycle counter register. Possible values are:

| C | Meaning |
|-----|---|
| 0b0 | When read, means the cycle counter is disabled. When written, has no effect. |
| 0b1 | When read, means the cycle counter is enabled. When written, enables the cycle counter. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter enable bit for [PMEVCNTR<n>_EL0](#).

If [PMCFGR.N](#) is less than 31, bits [30:[PMCFGR.N](#)] are RAZ/WI.

| P<n> | Meaning |
|------|---|
| 0b0 | When read, means that PMEVCNTR<n>_EL0 is disabled. When written, has no effect. |
| 0b1 | When read, means that PMEVCNTR<n>_EL0 event counter is enabled. When written, enables PMEVCNTR<n>_EL0 . |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMCNTENSET_ELO

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCNTENSET_ELO can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------------|
| PMU | 0xC00 | PMCNTENSET_ELO |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMCR_EL0, Performance Monitors Control Register

The PMCR_EL0 characteristics are:

Purpose

Provides details of the Performance Monitors implementation, including the number of counters implemented, and configures and controls the counters.

Configuration

External register PMCR_EL0 bits [7:0] are architecturally mapped to AArch32 System register [PMCR\[7:0\]](#).

External register PMCR_EL0 bits [7:0] are architecturally mapped to AArch64 System register [PMCR_EL0\[7:0\]](#).

PMCR_EL0 is in the Core power domain.

This register is only partially mapped to the internal [PMCR](#) System register. An external agent must use other means to discover the information held in [PMCR\[31:11\]](#), such as accessing [PMCFGR](#) and the ID registers.

Attributes

PMCR_EL0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|----|----|----|----|----|----|----|----|----|------|----|------|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RAZ/WI | | | | | | | | | | | RES0 | FZ | RES0 | LP | LC | DP | X | D | C | P | E | | | | | | | | | | |

Bits [31:11]

Reserved, RAZ/WI.

Hardware must implement this field as RAZ/WI. Software must not rely on the register reading as zero, and must use a read-modify-write sequence to write to the register.

Bit [10]

Reserved, RES0.

FZO, bit [9]

When FEAT_PMUv3p7 is implemented:

Freeze-on-overflow. Stop event counters on overflow.

In the description of this field:

- If EL2 is implemented and is using AArch32, PMN is [HDCR](#).HPMN.
- If EL2 is implemented and is using AArch64, PMN is [MDCR_EL2](#).HPMN.
- If EL2 is not implemented, PMN is PMCR_EL0.N.

| FZO | Meaning |
|-----|---|
| 0b0 | Do not freeze on overflow. |
| 0b1 | Event counter PMEVCNTR<n>_EL0 does not count when PMOVSCLR_EL0 [(PMN-1):0] is nonzero and n is in the range of affected event counters. |

If PMN is not 0, this field affects the operation of event counters in the range [0 .. (PMN-1)].

This field does not affect the operation of other event counters and [PMCCNTR_EL0](#).

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [8]

Reserved, RES0.

LP, bit [7]

When FEAT_PMUv3p5 is implemented:

Long event counter enable. Determines when unsigned overflow is recorded by an event counter overflow bit.

In the description of this field:

- If EL2 is implemented and is using AArch32, PMN is [HDCR](#).HPMN.
- If EL2 is implemented and is using AArch64, PMN is [MDCR_EL2](#).HPMN.
- If EL2 is not implemented, PMN is PMCR_EL0.N.

| LP | Meaning |
|-----|--|
| 0b0 | Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0 [31:0]. |
| 0b1 | Event counter overflow on increment that causes unsigned overflow of PMEVCNTR<n>_EL0 [63:0]. |

If PMN is not 0, this bit affects the operation of event counters in the range [0 .. (PMN-1)].

The field does not affect the operation of other event counters and [PMCCNTR_EL0](#).

The operation of this field applies even when EL2 is disabled in the current Security state.

Otherwise:

Reserved, RES0.

LC, bit [6]

When AArch32 is supported:

Long cycle counter enable. Determines when unsigned overflow is recorded by the cycle counter overflow bit.

| LC | Meaning |
|-----|--|
| 0b0 | Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0 [31:0]. |
| 0b1 | Cycle counter overflow on increment that causes unsigned overflow of PMCCNTR_EL0 [63:0]. |

Arm deprecates use of [PMCR_EL0](#).LC = 0.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES1.

DP, bit [5]

When EL3 is implemented or (FEAT_PMUv3p1 is implemented and EL2 is implemented):

Disable cycle counter when event counting is prohibited. The possible values of this bit are:

| DP | Meaning |
|-----|---|
| 0b0 | Cycle counting by PMCCNTR_EL0 is not affected by this mechanism. |
| 0b1 | Cycle counting by PMCCNTR_EL0 is disabled in prohibited regions: <ul style="list-style-type: none"> If FEAT_PMUv3p1 is implemented, EL2 is implemented, and MDCR_EL2.HPMD is 1, then cycle counting by PMCCNTR_EL0 is disabled at EL2. If FEAT_PMUv3p7 is implemented, EL3 is implemented and using AArch64, and MDCR_EL3.MPMX is 1, then cycle counting by PMCCNTR_EL0 is disabled at EL3. If EL3 is implemented, MDCR_EL3.SPME is 0, and either FEAT_PMUv3p7 is not implemented or MDCR_EL3.MPMX is 0, then cycle counting by PMCCNTR_EL0 is disabled at EL3 and in Secure state. <p>If MDCR_EL2.HPMN is not 0, this is when event counting by event counters in the range [0..(MDCR_EL2.HPMN-1)] is prohibited.</p> |

For more information, see 'Prohibiting event counting'.

The reset behavior of this field is:

- On a Warm reset:
 - When the implementation only supports execution in AArch32 state, this field resets to 0.
 - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

X, bit [4]

When the implementation includes a PMU event export bus:

Enable export of events in an IMPLEMENTATION DEFINED PMU event export bus.

| X | Meaning |
|-----|-------------------------------------|
| 0b0 | Do not export events. |
| 0b1 | Export events where not prohibited. |

This field enables the exporting of events over an IMPLEMENTATION DEFINED PMU event export bus to another device, for example to an OPTIONAL PE trace unit.

No events are exported when counting is prohibited.

This field does not affect the generation of Performance Monitors overflow interrupt requests or signaling to a cross-trigger interface (CTI) that can be implemented as signals exported from the PE.

When this register has an architecturally-defined reset value, if this field is implemented as an RW field it resets to:

- A value that is architecturally UNKNOWN if the reset is into an Exception level that is using AArch64.
- 0 if the reset is into an Exception level that is using AArch32.

Otherwise:

Reserved, RAZ/WI.

D, bit [3]**When AArch32 is supported:**

Clock divider.

| D | Meaning |
|-----|--|
| 0b0 | When enabled, PMCCNTR_EL0 counts every clock cycle. |
| 0b1 | When enabled, PMCCNTR_EL0 counts once every 64 clock cycles. |

If PMCR_EL0.LC == 1, this bit is ignored and the cycle counter counts every clock cycle.

Arm deprecates use of PMCR_EL0.D = 1.

When this register has an architecturally-defined reset value, if this field is implemented as an RW field it resets to:

- A value that is architecturally UNKNOWN if the reset is into an Exception level that is using AArch64.
- 0 if the reset is into an Exception level that is using AArch32.

Otherwise:

Reserved, RES0.

C, bit [2]

Cycle counter reset. The effects of writing to this bit are:

| C | Meaning |
|-----|--|
| 0b0 | No action. |
| 0b1 | Reset PMCCNTR_EL0 to zero. |

Note

Resetting [PMCCNTR_EL0](#) does not change the cycle counter overflow bit. If FEAT_PMUv3p5 is implemented, the value of PMCR_EL0.LC is ignored, and bits [63:0] of the cycle counter are reset.

Access to this field is **WO/RAZ**.

P, bit [1]

Event counter reset. The effects of writing to this bit are:

| P | Meaning |
|-----|--|
| 0b0 | No action. |
| 0b1 | Reset all event counters, not including PMCCNTR_EL0 , to zero. |

Note

Resetting the event counters does not change the event counter overflow bits. If FEAT_PMUv3p5 is implemented, the value of [MDCR_EL2.HLP](#), or PMCR_EL0.LP is ignored and bits [63:0] of all affected event counters are reset.

Access to this field is **WO/RAZ**.

E, bit [0]

Enable.

In the description of this field:

- If EL2 is implemented and is using AArch32, PMN is [HDCR](#).HPMN.
- If EL2 is implemented and is using AArch64, PMN is [MDCR_EL2](#).HPMN.
- If EL2 is not implemented, PMN is PMCR_EL0.N.

| E | Meaning |
|-----|--|
| 0b0 | PMCCNTR_EL0 is disabled and event counters PMEVCNTR<n>_EL0 , where n is in the range of affected event counters, are disabled. |
| 0b1 | PMCCNTR_EL0 and event counters PMEVCNTR<n>_EL0 , where n is in the range of affected event counters, are enabled by PMCNTENSET_EL0 . |

If PMN is not 0, this field affects the operation of event counters in the range [0 .. (PMN-1)].

This field does not affect the operation of other event counters.

The operation of this field applies even when EL2 is disabled in the current Security state.

The reset behavior of this field is:

- On a Warm reset, this field resets to 0.

Accessing PMCR_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMCR_EL0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xE04 | PMCR_EL0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

PMDEVAFF0, Performance Monitors Device Affinity register 0

The PMDEVAFF0 characteristics are:

Purpose

Copy of the low half of the PE [MPIDR_EL1](#) register that allows a debugger to determine which PE in a multiprocessor system the Performance Monitor component relates to.

Configuration

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required if the external interface to the PMU is implemented.

Attributes

PMDEVAFF0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------------------------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | MPIDR_EL1 lo | | | | | | | | | | | | | | | |

MPIDR_EL1lo, bits [31:0]

[MPIDR_EL1](#) low half. Read-only copy of the low half of [MPIDR_EL1](#), as seen from the highest implemented Exception level.

Accessing PMDEVAFF0

PMDEVAFF0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| PMU | 0xFA8 | PMDEVAFF0 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMDEVAFF1, Performance Monitors Device Affinity register 1

The PMDEVAFF1 characteristics are:

Purpose

Copy of the high half of the PE [MPIDR_EL1](#) register that allows a debugger to determine which PE in a multiprocessor system the Performance Monitor component relates to.

Configuration

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required if the external interface to the PMU is implemented.

Attributes

PMDEVAFF1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------------------------|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | | MPIDR_EL1hi | | | | | | | | | | | | | | |

MPIDR_EL1hi, bits [31:0]

[MPIDR_EL1](#) high half. Read-only copy of the high half of [MPIDR_EL1](#), as seen from the highest implemented Exception level.

Accessing PMDEVAFF1

PMDEVAFF1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| PMU | 0xFAC | PMDEVAFF1 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMDEVARCH, Performance Monitors Device Architecture register

The PMDEVARCH characteristics are:

Purpose

Identifies the programmers' model architecture of the Performance Monitor component.

Configuration

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

PMDEVARCH is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|---------|----------|----|----|----|---------|----|----|----|----------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ARCHITECT | | | | | | | | | | | PRESENT | REVISION | | | | ARCHVER | | | | ARCHPART | | | | | | | | | | | |

ARCHITECT, bits [31:21]

Defines the architecture of the component. For Performance Monitors, this is Arm Limited.

Bits [31:28] are the JEP106 continuation code, 0x4.

Bits [27:21] are the JEP106 ID code, 0x3B.

Reads as 0b01000111011.

Access to this field is **RO**.

PRESENT, bit [20]

Indicates that the DEVARCH is present.

Reads as 0b1.

Access to this field is **RO**.

REVISION, bits [19:16]

Defines the architecture revision. For architectures defined by Arm this is the minor revision.

For Performance Monitors, the revision defined by Armv8 is 0x0.

All other values are reserved.

Reads as 0b0000.

Access to this field is **RO**.

ARCHVER, bits [15:12]

Architecture Version. Defines the architecture version of the component.

All other values are reserved.

PMDEVARCH.ARCHVER and PMDEVARCH.ARCHPART are also defined as a single field, PMDEVARCH.ARCHID, so that PMDEVARCH.ARCHVER is PMDEVARCH.ARCHID[15:12].

Reads as 0b0010.

Access to this field is **RO**.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

| ARCHPART | Meaning | Applies when |
|----------|---|--------------|
| 0xA16 | Armv8-A PE performance monitors. | |
| 0xA26 | Armv8-A PE performance monitors, including the 64-bit programmers' model extension. | From Armv8.8 |

PMDEVARCH.ARCHVER and PMDEVARCH.ARCHPART are also defined as a single field, PMDEVARCH.ARCHID, so that PMDEVARCH.ARCHPART is PMDEVARCH.ARCHID[11:0].

Access to this field is **RO**.

Accessing PMDEVARCH

PMDEVARCH can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| PMU | 0xFBC | PMDEVARCH |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMDEVID, Performance Monitors Device ID register

The PMDEVID characteristics are:

Purpose

Provides information about features of the Performance Monitors implementation.

Configuration

If FEAT_DoPD is implemented, this register is in the Core power domain.

If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required from Armv8.2 and in any implementation that includes FEAT_PCSRv8p2. Otherwise, its location is RES0.

Note

Before Armv8.2, the PC Sample-based Profiling Extension can be implemented in the external debug register space, as indicated by the value of [EDDEVID.PCSample](#).

Attributes

PMDEVID is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|----------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | PCSample | | | |

Bits [31:4]

Reserved, RES0.

PCSample, bits [3:0]

Indicates the level of PC Sample-based Profiling support using Performance Monitors registers.

| PCSample | Meaning |
|----------|--|
| 0b0000 | PC Sample-based Profiling Extension is not implemented in the Performance Monitors register space. |
| 0b0001 | PC Sample-based Profiling Extension is implemented in the Performance Monitors register space. |

All other values are reserved.

FEAT_PCSRv8p2 implements the functionality identified by the value 0b0001.

Accessing PMDEVID

PMDEVID can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
|-----------|--------|----------|

| | | |
|-----|-------|---------|
| PMU | 0xFC8 | PMDEVID |
|-----|-------|---------|

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMDEVTYPE, Performance Monitors Device Type register

The PMDEVTYPE characteristics are:

Purpose

Indicates to a debugger that this component is part of a PE's performance monitor interface.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Attributes

PMDEVTYPE is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-----|---|---|---|-------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | SUB | | | | MAJOR | | | |

Bits [31:8]

Reserved, RES0.

SUB, bits [7:4]

Subtype. Indicates this is a component within a PE.

Reads as 0b0001.

Access to this field is **RO**.

MAJOR, bits [3:0]

Major type. Indicates this is a performance monitor component.

Reads as 0b0110.

Access to this field is **RO**.

Accessing PMDEVTYPE

PMDEVTYPE can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| PMU | 0xFCC | PMDEVTYPE |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.

- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMEVCNTR<n>_EL0, Performance Monitors Event Count Registers, n = 0 - 30

The PMEVCNTR<n>_EL0 characteristics are:

Purpose

Holds event counter n, which counts events, where n is 0 to 30.

Configuration

External register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to AArch64 System register [PMEVCNTR<n>_EL0\[31:0\]](#).

External register PMEVCNTR<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMEVCNTR<n>\[31:0\]](#).

PMEVCNTR<n>_EL0 is in the Core power domain.

Attributes

PMEVCNTR<n>_EL0 is a:

- 64-bit register when FEAT_PMUv3p5 is implemented
- 32-bit register otherwise

Field descriptions

When FEAT_PMUv3p5 is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| Event counter n | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| Event counter n | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

Bits [63:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

If the highest implemented Exception level is using AArch32, the optional external interface to the performance monitors is implemented, and the [PMCR](#).LP and [HDCR](#).HLP bits are RAZ/WI, then locations in the external interface to the performance monitors that map to PMEVCNTR<n>_EL0[63:32] return UNKNOWN values on reads.

If the implementation does not support AArch64, bits [63:32] of the event counters are not required to be implemented.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| Event counter n | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:0]

Event counter n. Value of event counter n, where n is the number of this register and is a number from 0 to 30.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMEVCNTR<n>_EL0

External accesses to the performance monitors ignore [PMUSERENR_EL0](#) and, if implemented, [MDCR_EL2](#).{TPM, TPMCR, HPMN} and [MDCR_EL3](#).TPM. This means that all counters are accessible regardless of the current Exception level or privilege of the access.

If FEAT_PMuV3p5 is not implemented, when IsCorePowered(), DoubleLockStatus(), OSLockStatus() or !AllowExternalPMUAccess(), 32-bit accesses to 0x004+8×n have a CONSTRAINED UNPREDICTABLE behavior.

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMEVCNTR<n>_EL0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|-----------------|
| PMU | 0x000 + (8 * n) | PMEVCNTR<n>_EL0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

PMEVFILTR<n>, Performance Monitors Event Type Select Register <n>, n = 0 - 30

The PMEVFILTR<n> characteristics are:

Purpose

External access to [PMEVTYPER<n>_EL0](#)[63:32].

Configuration

External register PMEVFILTR<n> bits [31:0] are architecturally mapped to AArch64 System register [PMEVTYPER<n>_EL0](#)[63:32] when AArch64 is supported.

PMEVFILTR<n> is in the Core power domain.

This register is present only when FEAT_PMUv3_TH is implemented. Otherwise, direct accesses to PMEVFILTR<n> are RES0.

Note

If FEAT_Debugv8p4 is implemented, the OPTIONAL Software Lock is not implemented.

If FEAT_DoPD is implemented, FEAT_DoubleLock is not implemented.

Attributes

PMEVFILTR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| TC | | | RES0 | | | | | | | | | | | | | | | | | | TH | | | | | | | | | | |

TC, bits [31:29]
When FEAT_PMUv3_TH is implemented:

Threshold Control. Defines the threshold function. In the description of this field, the value V is the value the event specified by [PMEVTYPER<n>_EL0](#) would increment the counter by on a processor cycle if the threshold function is disabled. Comparisons treat V and PMEVFILTR<n>.TH as unsigned integer values.

| TC | Meaning |
|-------|--|
| 0b000 | Not-equal. The counter increments by V on each processor cycle when V is not equal to PMEVFILTR<n>.TH. If PMEVFILTR<n>.TH is zero, the threshold function is disabled. |
| 0b001 | Not-equal, count. The counter increments by 1 on each processor cycle when V is not equal to PMEVFILTR<n>.TH. |
| 0b010 | Equals. The counter increments by V on each processor cycle when V is equal to PMEVFILTR<n>.TH. |
| 0b011 | Equals, count. The counter increments by 1 on each processor cycle when V is equal to PMEVFILTR<n>.TH. |
| 0b100 | Greater-than-or-equal. The counter increments by V on each processor cycle when V is PMEVFILTR<n>.TH or more. |
| 0b101 | Greater-than-or-equal, count. The counter increments by 1 on each processor cycle when V is PMEVFILTR<n>.TH or more. |
| 0b110 | Less-than. The counter increments by V on each processor cycle when V is less than PMEVFILTR<n>.TH. |
| 0b111 | Less-than, count. The counter increments by 1 on each processor cycle when V is less than PMEVFILTR<n>.TH. |

The reset behavior of this field is:

- On a Warm reset:
 - When AArch32 is supported, this field resets to 0.
 - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [28:12]

Reserved, RES0.

TH, bits [11:0]

When FEAT_PMUv3_TH is implemented:

Threshold value. Provides the unsigned value for the threshold function defined by PMEVFILTR<n>.TC.

If PMEVFILTR<n>.TC is 0b000 and PMEVFILTR<n>.TH is zero, then the threshold function is disabled.

If [PMMIR](#).THWIDTH is less than 12, then bits PMEVFILTR<n>.TH[11:[PMMIR](#).THWIDTH] are RES0. This accounts for the behavior when writing a value greater-than-or-equal-to $2^{(\text{PMMIR.THWIDTH})}$.

The reset behavior of this field is:

- On a Warm reset:
 - When AArch32 is supported, this field resets to 0.
 - Otherwise, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing PMEVFILTR<n>

PMEVFILTR<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|--------------|
| PMU | 0xA00 + (4 * n) | PMEVFILTR<n> |

This interface is accessible as follows:

- When DoubleLockStatus(), or !IsCorePowered(), or OSLockStatus() or !AllowExternalPMUAccess() accesses to this register generate an error response.
- When SoftwareLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMEVTYPER<n>_EL0, Performance Monitors Event Type Registers, n = 0 - 30

The PMEVTYPER<n>_EL0 characteristics are:

Purpose

Configures event counter n, where n is 0 to 30.

Configuration

External register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to AArch64 System register [PMEVTYPER<n>_EL0\[31:0\]](#).

External register PMEVTYPER<n>_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMEVTYPER<n>\[31:0\]](#).

PMEVTYPER<n>_EL0 is in the Core power domain.

If event counter n is not implemented:

- When IsCorePowered() && !DoubleLockStatus() && !OSLockStatus() && AllowExternalPMUAccess(), accesses are RES0.
- Otherwise, it is CONSTRAINED UNPREDICTABLE whether accesses to this register are RES0 or generate an error response.

Attributes

PMEVTYPER<n>_EL0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|-----|-----|-----|----|----|----|----|-----|-----|-----|------|----|----|----|-----------------|----|----|----|----|----|---------------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| P | U | NSK | NSU | NSH | M | MT | SH | T | RLK | RLU | RLH | RES0 | | | | evtCount[15:10] | | | | | | evtCount[9:0] | | | | | | | | | |

P, bit [31]

Privileged filtering bit. Controls counting in EL1.

If EL3 is implemented, then counting in Non-secure EL1 is further controlled by the PMEVTYPER<n>_EL0.NSK bit.

If FEAT_RME is implemented, then counting in Realm EL1 is further controlled by the PMEVTYPER<n>_EL0.RLK bit.

| P | Meaning |
|-----|-----------------------------|
| 0b0 | Count events in EL1. |
| 0b1 | Do not count events in EL1. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

U, bit [30]

User filtering bit. Controls counting in EL0.

If EL3 is implemented, then counting in Non-secure EL0 is further controlled by the PMEVTYPER<n>_EL0.NSU bit.

If FEAT_RME is implemented, then counting in Realm EL0 is further controlled by the PMEVTYPER<n>_EL0.RLU bit.

| U | Meaning |
|-----|-----------------------------|
| 0b0 | Count events in EL0. |
| 0b1 | Do not count events in EL0. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

NSK, bit [29]

When EL3 is implemented:

Non-secure EL1 (kernel) modes filtering bit. Controls counting in Non-secure EL1.

If the value of this bit is equal to the value of the PMEVTYPER<n>_EL0.P bit, events in Non-secure EL1 are counted.

Otherwise, events in Non-secure EL1 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSU, bit [28]

When EL3 is implemented:

Non-secure EL0 (Unprivileged) filtering bit. Controls counting in Non-secure EL0.

If the value of this bit is equal to the value of the PMEVTYPER<n>_EL0.U bit, events in Non-secure EL0 are counted.

Otherwise, events in Non-secure EL0 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSH, bit [27]

When EL2 is implemented:

EL2 (Hypervisor) filtering bit. Controls counting in EL2.

If FEAT_SEL2 and EL3 are implemented, counting in Secure EL2 is further controlled by the PMEVTYPER<n>_EL0.SH bit.

If FEAT_RME is implemented, then counting in Realm EL2 is further controlled by the PMEVTYPER<n>_EL0.RLH bit.

| NSH | Meaning |
|-----|-----------------------------|
| 0b0 | Do not count events in EL2. |
| 0b1 | Count events in EL2. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

M, bit [26]

When EL3 is implemented:

EL3 filtering bit.

If the value of this bit is equal to the value of the PMEVTYPER<n>_EL0.P bit, events in EL3 are counted.

Otherwise, events in EL3 are not counted.

Most applications can ignore this field and set its value to 0b0.

Note

This field is not visible in the AArch32 [PMEVTYPER<n>](#) System register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

MT, bit [25]

When (FEAT_MTPMU is implemented and enabled) or an IMPLEMENTATION DEFINED multi-threaded PMU Extension is implemented:

Multithreading.

| MT | Meaning |
|-----|--|
| 0b0 | Count events only on controlling PE. |
| 0b1 | Count events from any PE with the same affinity at level 1 and above as this PE. |

Note

- When the lowest level of affinity consists of logical PEs that are implemented using a multi-threading type approach, an implementation is described as multi-threaded. That is, the performance of PEs at the lowest affinity level is highly interdependent.
- Events from a different thread of a multithreaded implementation are not Attributable to the thread counting the event.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

SH, bit [24]**When FEAT_SEL2 is implemented and EL3 is implemented:**

Secure EL2 filtering.

If the value of this bit is not equal to the value of the PMEVTYPER<n>_EL0.NSH bit, events in Secure EL2 are counted.

Otherwise, events in Secure EL2 are not counted.

Note

This field is not visible in the AArch32 [PMEVTYPER<n>](#) System register.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

T, bit [23]**When FEAT_TME is implemented:**

Transactional state filtering bit. Controls counting in Transactional state.

| T | Meaning |
|-----|--|
| 0b0 | This bit has no effect on the filtering of events. |
| 0b1 | Do not count events in Transactional state. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLK, bit [22]**When FEAT_RME is implemented:**

Realm EL1 (kernel) filtering bit. Controls counting in Realm EL1.

If the value of this bit is equal to the value of the PMEVTYPER<n>_EL0.P bit, events in Realm EL1 are counted.

Otherwise, events in Realm EL1 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLU, bit [21]**When FEAT_RME is implemented:**

Realm EL0 (unprivileged) filtering bit. Controls counting in Realm EL0.

If the value of this bit is equal to the value of the PMEVTYPER<n>_EL0.U bit, events in Realm EL0 are counted.

Otherwise, events in Realm EL0 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

RLH, bit [20]

When FEAT_RME is implemented:

Realm EL2 filtering bit. Controls counting in Realm EL2.

If the value of this bit is not equal to the value of the PMEVTYPER<n>_EL0.NSH bit, events in Realm EL2 are counted.

Otherwise, events in Realm EL2 are not counted.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [19:16]

Reserved, RES0.

evtCount[15:10], bits [15:10]

When FEAT_PMUv3p1 is implemented:

Extension to evtCount[9:0]. For more information, see evtCount[9:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

evtCount[9:0], bits [9:0]

Event to count. The event number of the event that is counted by event counter [PMEVCNTR<n>_EL0](#).

Software must program this field with an event that is supported by the PE being programmed.

The ranges of event numbers allocated to each type of event are shown in 'Allocation of the PMU event number space'.

If FEAT_PMUv3p8 is implemented and PMEVTYPER<n>_EL0.evtCount is programmed to an event that is reserved or not supported by the PE, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.

Otherwise, if PMEVTYPER<n>_EL0.evtCount is programmed to an event that is reserved or not supported by the PE, the behavior depends on the value written:

- For the range 0x0000 to 0x003F, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.
- If FEAT_PMUv3p1 is implemented, for the range 0x4000 to 0x403F, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.
- For other values, it is UNPREDICTABLE what event, if any, is counted, and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is UNKNOWN.

Note

UNPREDICTABLE means the event must not expose privileged information.

Arm recommends that for all values that represent reserved or unsupported events, no events are counted and the value returned by a direct or external read of the PMEVTYPER<n>_EL0.evtCount field is the value written to the field.

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMEVTYPER<n>_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMEVTYPER<n>_EL0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|------------------|
| PMU | 0x400 + (4 * n) | PMEVTYPER<n>_EL0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMINTENCLR_EL1, Performance Monitors Interrupt Enable Clear register

The PMINTENCLR_EL1 characteristics are:

Purpose

Disables the generation of interrupt requests on overflows from the Cycle Count Register, [PMCCNTR_EL0](#), and the event counters [PMEVCNTR<n>_EL0](#). Reading the register shows which overflow interrupt requests are enabled.

Configuration

External register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch64 System register [PMINTENCLR_EL1\[31:0\]](#).

External register PMINTENCLR_EL1 bits [31:0] are architecturally mapped to AArch32 System register [PMINTENCLR\[31:0\]](#).

PMINTENCLR_EL1 is in the Core power domain.

Attributes

PMINTENCLR_EL1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

C, bit [31]

[PMCCNTR_EL0](#) overflow interrupt request disable bit. Possible values are:

| C | Meaning |
|-----|--|
| 0b0 | When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect. |
| 0b1 | When read, means the cycle counter overflow interrupt request is enabled. When written, disables the cycle count overflow interrupt request. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter overflow interrupt request disable bit for [PMEVCNTR<n>_EL0](#).

If [PMCFGR.N](#) is less than 31, bits [30:[PMCFGR.N](#)] are RAZ/WI.

| P<n> | Meaning |
|------|---|
| 0b0 | When read, means that the PMEVCNTR<n>_EL0 event counter interrupt request is disabled. When written, has no effect. |
| 0b1 | When read, means that the PMEVCNTR<n>_EL0 event counter interrupt request is enabled. When written, disables the PMEVCNTR<n>_EL0 interrupt request. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMINTENCLR_EL1

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMINTENCLR_EL1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------------|
| PMU | 0xC60 | PMINTENCLR_EL1 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMINTENSET_EL1, Performance Monitors Interrupt Enable Set register

The PMINTENSET_EL1 characteristics are:

Purpose

Enables the generation of interrupt requests on overflows from the Cycle Count Register, [PMCCNTR_EL0](#), and the event counters [PMEVCNTR<n>_EL0](#). Reading the register shows which overflow interrupt requests are enabled.

Configuration

External register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch64 System register [PMINTENSET_EL1\[31:0\]](#).

External register PMINTENSET_EL1 bits [31:0] are architecturally mapped to AArch32 System register [PMINTENSET\[31:0\]](#).

PMINTENSET_EL1 is in the Core power domain.

Attributes

PMINTENSET_EL1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

C, bit [31]

[PMCCNTR_EL0](#) overflow interrupt request enable bit. Possible values are:

| C | Meaning |
|-----|---|
| 0b0 | When read, means the cycle counter overflow interrupt request is disabled. When written, has no effect. |
| 0b1 | When read, means the cycle counter overflow interrupt request is enabled. When written, enables the cycle count overflow interrupt request. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter overflow interrupt request enable bit for [PMEVCNTR<n>_EL0](#).

If [PMCFGR.N](#) is less than 31, bits [30:[PMCFGR.N](#)] are RAZ/WI.

| P<n> | Meaning |
|------|--|
| 0b0 | When read, means that the PMEVCNTR<n>_EL0 event counter interrupt request is disabled. When written, has no effect. |
| 0b1 | When read, means that the PMEVCNTR<n>_EL0 event counter interrupt request is enabled. When written, enables the PMEVCNTR<n>_EL0 interrupt request. |

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMINTENSET_EL1

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMINTENSET_EL1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------------|
| PMU | 0xC40 | PMINTENSET_EL1 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMITCTRL, Performance Monitors Integration mode Control register

The PMITCTRL characteristics are:

Purpose

Enables the Performance Monitors to switch from default mode into integration mode, where test software can control directly the inputs and outputs of the PE, for integration testing or topology detection.

Configuration

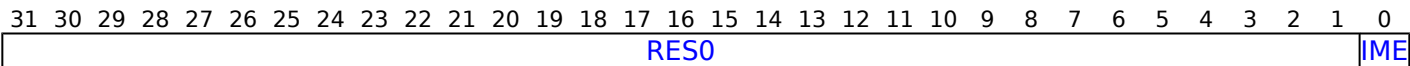
It is IMPLEMENTATION DEFINED whether PMITCTRL is implemented in the Core power domain or in the Debug power domain.

Implementation of this register is OPTIONAL.

Attributes

PMITCTRL is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

IME, bit [0]

Integration mode enable. When IME == 1, the device reverts to an integration mode to enable integration testing or topology detection. The integration mode behavior is IMPLEMENTATION DEFINED.

| IME | Meaning |
|-----|---------------------------|
| 0b0 | Normal operation. |
| 0b1 | Integration mode enabled. |

The following resets apply:

- If the register is implemented in the Core power domain:
 - On a Cold reset, this field resets to 0.
 - On an External debug reset, the value of this field is unchanged.
 - On a Warm reset, the value of this field is unchanged.
- If the register is implemented in the External debug power domain:
 - On a Cold reset, the value of this field is unchanged.
 - On an External debug reset, this field resets to 0.
 - On a Warm reset, the value of this field is unchanged.

Accessing PMITCTRL

PMITCTRL can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xF00 | PMITCTRL |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register are **IMPDEF**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMLAR, Performance Monitors Lock Access Register

The PMLAR characteristics are:

Purpose

Allows or disallows access to the Performance Monitors registers through a memory-mapped interface.

The optional Software Lock provides a lock to prevent memory-mapped writes to the Performance Monitors registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the Performance Monitors registers. It does not, and cannot, prevent all accidental or malicious damage.

Configuration

If FEAT_DoPD is implemented, Software Lock is not implemented by the architecturally-defined debug components of the PE in the Core power domain.

If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Software uses PMLAR to set or clear the lock, and [PMLSR](#) to check the current status of the lock.

Attributes

PMLAR is a 32-bit register.

Field descriptions

When Software Lock is implemented:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | KEY | | | | | | | | | | | | | | | |

KEY, bits [31:0]

Lock Access control. Writing the key value 0xC5ACCE55 to this field unlocks the lock, enabling write accesses to this component's registers through a memory-mapped interface.

Writing any other value to this register locks the lock, disabling write accesses to this component's registers through a memory mapped interface.

Otherwise:

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |

Otherwise

Bits [31:0]

Reserved, RES0.

Accessing PMLAR

PMLAR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xFB0 | PMLAR |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **WO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMLSR, Performance Monitors Lock Status Register

The PMLSR characteristics are:

Purpose

Indicates the current status of the software lock for Performance Monitors registers.

The optional Software Lock provides a lock to prevent memory-mapped writes to the Performance Monitors registers. Use of this lock mechanism reduces the risk of accidental damage to the contents of the Performance Monitors registers. It does not, and cannot, prevent all accidental or malicious damage.

Configuration

If FEAT_DoPD is implemented, Software Lock is not implemented by the architecturally-defined debug components of the PE in the Core power domain.

If FEAT_DoPD is not implemented, this register is in the Debug power domain.

Software uses [PMLAR](#) to set or clear the lock, and PMLSR to check the current status of the lock.

Attributes

PMLSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|-----|-----|-----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | nTT | SLK | SLI |

Bits [31:3]

Reserved, RES0.

nTT, bit [2]

Not thirty-two bit access required.

Reads as 0b0.

Access to this field is **RO**.

SLK, bit [1]

When Software Lock is implemented and FEAT_DoPD is not implemented:

Software Lock status for this component. For an access to LSR that is not a memory-mapped access, or when Software Lock is not implemented, this field is RES0.

For memory-mapped accesses when Software Lock is implemented, possible values of this field are:

| SLK | Meaning |
|-----|---|
| 0b0 | Lock clear. Writes are permitted to this component's registers. |
| 0b1 | Lock set. Writes to this component's registers are ignored, and reads have no side effects. |

The reset behavior of this field is:

- On an External debug reset, this field resets to 1.

Otherwise:

Reserved, RAZ.

SLI, bit [0]

Software Lock implemented. For an access to LSR that is not a memory-mapped access, this field is RAZ. For memory-mapped accesses, the value of this field is IMPLEMENTATION DEFINED. Permitted values are:

| SLI | Meaning |
|-----|--|
| 0b0 | Software Lock not implemented or not memory-mapped access. |
| 0b1 | Software Lock implemented and memory-mapped access. |

Accessing PMLSR

PMLSR can be accessed through the memory-mapped interfaces:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xFB4 | PMLSR |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMMIR, Performance Monitors Machine Identification Register

The PMMIR characteristics are:

Purpose

Describes Performance Monitors parameters specific to the implementation.

Configuration

PMMIR is in the Core power domain.

This register is present only when FEAT_PMUv3p4 is implemented. Otherwise, direct accesses to PMMIR are RES0.

Attributes

PMMIR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|---------|----|----|----|-----------|----|----|----|-----------|----|----|----|----|----|-------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | THWIDTH | | | | BUS_WIDTH | | | | BUS_SLOTS | | | | | | SLOTS | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

THWIDTH, bits [23:20]

[PMEVFILTR<n>_TH](#) width. Indicates implementation of the FEAT_PMUv3_TH feature, and, if implemented, the size of the [PMEVFILTR<n>_TH](#) field.

| THWIDTH | Meaning |
|---------|--|
| 0b0000 | FEAT_PMUv3_TH is not implemented. |
| 0b0001 | 1 bit. PMEVFILTR<n>_TH [11:1] are RES0. |
| 0b0010 | 2 bits. PMEVFILTR<n>_TH [11:2] are RES0. |
| 0b0011 | 3 bits. PMEVFILTR<n>_TH [11:3] are RES0. |
| 0b0100 | 4 bits. PMEVFILTR<n>_TH [11:4] are RES0. |
| 0b0101 | 5 bits. PMEVFILTR<n>_TH [11:5] are RES0. |
| 0b0110 | 6 bits. PMEVFILTR<n>_TH [11:6] are RES0. |
| 0b0111 | 7 bits. PMEVFILTR<n>_TH [11:7] are RES0. |
| 0b1000 | 8 bits. PMEVFILTR<n>_TH [11:8] are RES0. |
| 0b1001 | 9 bits. PMEVFILTR<n>_TH [11:9] are RES0. |
| 0b1010 | 10 bits. PMEVFILTR<n>_TH [11:10] are RES0. |
| 0b1011 | 11 bits. PMEVFILTR<n>_TH [11] is RES0. |
| 0b1100 | 12 bits. |

All other values are reserved.

If FEAT_PMUv3_TH is not implemented, this field is zero.

Otherwise, the largest value that can be written to [PMEVFILTR<n>_TH](#) is $2^{(\text{PMMIR.THWIDTH})}$ minus one.

Access to this field is **RO**.

BUS_WIDTH, bits [19:16]

Bus width. Indicates the number of bytes each BUS_ACCESS event relates to. Encoded as $\text{Log}_2(\text{number of bytes})$, plus one. Defined values are:

| BUS_WIDTH | Meaning |
|------------------|-----------------------------------|
| 0b0000 | The information is not available. |
| 0b0011 | Four bytes. |
| 0b0100 | 8 bytes. |
| 0b0101 | 16 bytes. |
| 0b0110 | 32 bytes. |
| 0b0111 | 64 bytes. |
| 0b1000 | 128 bytes. |
| 0b1001 | 256 bytes. |
| 0b1010 | 512 bytes. |
| 0b1011 | 1024 bytes. |
| 0b1100 | 2048 bytes. |

All other values are reserved.

Each transfer is up to this number of bytes. An access might be smaller than the bus width.

When this field is nonzero, each access counted by BUS_ACCESS is at most BUS_WIDTH bytes. An implementation might treat a wide bus as multiple narrower buses, such that a wide access on the bus increments the BUS_ACCESS counter by more than one.

Access to this field is **RO**.

BUS_SLOTS, bits [15:8]

Bus count. The largest value by which the BUS_ACCESS event might increment by in a single BUS_CYCLES cycle.

When this field is nonzero, the largest value by which the BUS_ACCESS event might increment in a single BUS_CYCLES cycle is BUS_SLOTS.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

SLOTS, bits [7:0]

Operation width. The largest value by which the STALL_SLOT event might increment by in a single cycle. If the STALL_SLOT event is not implemented, this field might read as zero.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMMIR

If the Core power domain is off or in a low-power state, access on this interface returns an Error.

PMMIR can be accessed through the external debug interface:

| Component | Offset | Instance |
|------------------|---------------|-----------------|
| PMU | 0xE40 | PMMIR |

This interface is accessible as follows:

- When !IsCorePowered(), or DoubleLockStatus(), or OSLockStatus() or !AllowExternalPMUAccess() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

PMOVSLR_EL0, Performance Monitors Overflow Flag Status Clear register

The PMOVSLR_EL0 characteristics are:

Purpose

Contains the state of the overflow bit for the Cycle Count Register, [PMCCNTR_EL0](#), and each of the implemented event counters [PMEVCNTR<n>](#). Writing to this register clears these bits.

Configuration

External register PMOVSLR_EL0 bits [31:0] are architecturally mapped to AArch64 System register [PMOVSLR_EL0\[31:0\]](#).

External register PMOVSLR_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMOVSR\[31:0\]](#).

PMOVSLR_EL0 is in the Core power domain.

Attributes

PMOVSLR_EL0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

C, bit [31]

Cycle counter overflow clear bit.

| C | Meaning |
|-----|--|
| 0b0 | When read, means the cycle counter has not overflowed since this bit was last cleared. When written, has no effect. |
| 0b1 | When read, means the cycle counter has overflowed since this bit was last cleared. When written, clears the cycle counter overflow bit to 0. |

[PMCR_EL0](#).LC controls whether an overflow is detected from unsigned overflow of [PMCCNTR_EL0](#)[31:0] or unsigned overflow of [PMCCNTR_EL0](#)[63:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter overflow clear bit for [PMEVCNTR<n>_EL0](#).

If [PMCFGR](#).N is less than 31, bits [30:[PMCFGR](#).N] are RAZ/WI.

| P<n> | Meaning |
|------|---|
| 0b0 | When read, means that PMEVCNTR<n>_EL0 has not overflowed since this bit was last cleared. When written, has no effect. |
| 0b1 | When read, means that PMEVCNTR<n>_EL0 has overflowed since this bit was last cleared. When written, clears the PMEVCNTR<n>_EL0 overflow bit to 0. |

If FEAT_PMUv3p5 is implemented, [MDCR_EL2.HLP](#) and [PMCR_EL0.LP](#) control whether an overflow is detected from unsigned overflow of [PMEVCNTR<n>_EL0](#)[31:0] or unsigned overflow of [PMEVCNTR<n>_EL0](#)[63:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMOVSLR_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMOVSLR_EL0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-------------|
| PMU | 0xC80 | PMOVSLR_EL0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMOVSSET_EL0, Performance Monitors Overflow Flag Status Set register

The PMOVSSET_EL0 characteristics are:

Purpose

Sets the state of the overflow bit for the Cycle Count Register, [PMCCNTR_EL0](#), and each of the implemented event counters [PMEVCNTR<n>](#).

Configuration

External register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch64 System register [PMOVSSET_EL0\[31:0\]](#).

External register PMOVSSET_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMOVSSETI\[31:0\]](#).

PMOVSSET_EL0 is in the Core power domain.

Attributes

PMOVSSET_EL0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| C | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

C, bit [31]

Cycle counter overflow set bit.

| C | Meaning |
|-----|--|
| 0b0 | When read, means the cycle counter has not overflowed since this bit was last cleared. When written, has no effect. |
| 0b1 | When read, means the cycle counter has overflowed since this bit was last cleared. When written, sets the cycle counter overflow bit to 1. |

[PMCR_EL0](#).LC controls whether an overflow is detected from unsigned overflow of [PMCCNTR_EL0](#)[31:0] or unsigned overflow of [PMCCNTR_EL0](#)[63:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

P<n>, bit [n], for n = 30 to 0

Event counter overflow set bit for [PMEVCNTR<n>_EL0](#).

If [PMCFGR](#).N is less than 31, bits [30:[PMCFGR](#).N] are RAZ/WI.

| P<n> | Meaning |
|------|---|
| 0b0 | When read, means that PMEVCNTR<n>_EL0 has not overflowed since this bit was last cleared. When written, has no effect. |
| 0b1 | When read, means that PMEVCNTR<n>_EL0 has overflowed since this bit was last cleared. When written, sets the PMEVCNTR<n>_EL0 overflow bit to 1. |

If FEAT_PMUv3p5 is implemented, [MDCR_EL2.HLP](#) and [PMCR_EL0.LP](#) control whether an overflow is detected from unsigned overflow of [PMEVCNTR<n>_EL0](#)[31:0] or unsigned overflow of [PMEVCNTR<n>_EL0](#)[63:0].

The reset behavior of this field is:

- On a Warm reset, this field resets to an architecturally UNKNOWN value.

Accessing PMOVSSET_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMOVSSET_EL0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|--------------|
| PMU | 0xCC0 | PMOVSSET_EL0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **RO**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **RW**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMPCSR, Program Counter Sample Register

The PMPCSR characteristics are:

Purpose

Holds a sampled instruction address value.

Configuration

PMPCSR is in the Core power domain.

This register is present only when FEAT_PCSRv8p2 is implemented. Otherwise, direct accesses to PMPCSR are RES0.

Note

Before Armv8.2, the PC Sample-based Profiling Extension can be implemented in the external debug register space, as indicated by the value of [EDDEVID](#).PCSample.

Support for 64-bit atomic reads is IMPLEMENTATION DEFINED. If 64-bit atomic reads are implemented, a 64-bit read of PMPCSR has the same side-effect as a 32-bit read of PMCSR[31:0] followed by a 32-bit read of PMPCSR[63:32], returning the combined value. For example, if the PE is in Debug state then a 64-bit atomic read returns bits[31:0] == 0xFFFFFFFF and bits[63:32] UNKNOWN.

Attributes

PMPCSR is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------|----|----|-----|------|-----------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| NS | EL | T | NSE | RES0 | PCSample[55:32] | | | | | | | | | | | | | | | | | | | | | | | | | | |
| PCSample[31:0] | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

NS, bit [63]

When FEAT_RME is implemented:

Together with the NSE field, indicates the Security state that is associated with the most recent PMPCSR sample or, when it is read as a single atomic 64-bit read, the current PMPCSR sample.

| NSE | NS | Meaning |
|-----|-----|-------------|
| 0b0 | 0b0 | Secure. |
| 0b0 | 0b1 | Non-secure. |
| 0b1 | 0b0 | Root. |
| 0b1 | 0b1 | Realm. |

Otherwise:

Non-secure state sample. Indicates the Security state that is associated with the most recent PMPCSR sample or, when it is read as a single atomic 64-bit read, the current PMPCSR sample.

If EL3 is not implemented, this bit indicates the Effective value of SCR.NS.

| NS | Meaning |
|-----|----------------------------------|
| 0b0 | Sample is from Secure state. |
| 0b1 | Sample is from Non-secure state. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

EL, bits [62:61]

Exception level status sample. Indicates the Exception level that is associated with the most recent PMPCSR sample or, when it is read as a single atomic 64-bit read, the current PMPCSR sample.

| EL | Meaning |
|------|---------------------|
| 0b00 | Sample is from EL0. |
| 0b01 | Sample is from EL1. |
| 0b10 | Sample is from EL2. |
| 0b11 | Sample is from EL3. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

T, bit [60]

When FEAT_TME is implemented:

Transactional state of the sample. Indicates the Transactional state that is associated with the most recent PMPCSR sample or, when it is read as a single atomic 64-bit read, the current PMPCSR sample.

| T | Meaning |
|-----|---|
| 0b0 | Sample is from Non-transactional state. |
| 0b1 | Sample is from Transactional state. |

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

NSE, bit [59]

When FEAT_RME is implemented:

Together with the NS field, indicates the Security state that is associated with the most recent PMPCSR sample or, when it is read as a single atomic 64-bit read, the current PMPCSR sample.

For a description of the values derived by evaluating NS and NSE together, see PMPCSR.NS.

Otherwise:

Reserved, RES0.

Bits [58:56]

Reserved, RES0.

PCSample[55:32], bits [55:32]

Bits[55:32] of the sampled instruction address value. The translation regime that PMPCSR samples can be determined from PMPCSR.{NS,EL}.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

PCSample[31:0], bits [31:0]

Bits[31:0] of the sampled instruction address value.

PMPCSR[31:0] reads as 0xFFFFFFFF when any of the following are true:

- The PE is in Debug state.
- PC Sample-based profiling is prohibited.

If a branch instruction has retired since the PE left reset state, then the first read of PMPCSR[31:0] is permitted but not required to return 0xFFFFFFFF.

PMPCSR[31:0] reads as an UNKNOWN value when any of the following are true:

- The PE is in reset state.
- No branch instruction has retired since the PE left reset state, Debug state, or a state where PC Sample-based Profiling is prohibited.
- No branch instruction has retired since the last read of PMPCSR[31:0].

For the cases where a read of PMPCSR[31:0] returns 0xFFFFFFFF or an UNKNOWN value, the read has the side-effect of setting PMPCSR[63:32], [PMCID1SR](#), [PMCID2SR](#), and [PMVIDSR](#) to UNKNOWN values.

Otherwise, a read of PMPCSR[31:0] returns bits [31:0] of the sampled instruction address value and has the side-effect of indirectly writing to PMPCSR[63:32], [PMCID1SR](#), [PMCID2SR](#), and [PMVIDSR](#). The translation regime that PMPCSR samples can be determined from PMPCSR.{NS,EL}.

For a read of PMPCSR[31:0] from the memory-mapped interface, if PMLSR.SLK == 1, meaning the OPTIONAL Software Lock is locked, then the side-effect of the access does not occur and PMPCSR[63:32], [PMCID1SR](#), [PMCID2SR](#), and [PMVIDSR](#) are unchanged.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing PMPCSR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

PMPCSR can be accessed through the external debug interface:

| Component | Offset | Instance | Range |
|-----------|--------|----------|-------|
| PMU | 0x200 | PMPCSR | 31:0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

| Component | Offset | Instance | Range |
|-----------|--------|----------|-------|
| PMU | 0x204 | PMPCSR | 63:32 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

| Component | Offset | Instance | Range |
|-----------|--------|----------|-------|
| PMU | 0x220 | PMPCSR | 31:0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.

- Otherwise accesses to this register generate an error response.

| Component | Offset | Instance | Range |
|-----------|--------|----------|-------|
| PMU | 0x224 | PMPCSR | 63:32 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMPIDR0, Performance Monitors Peripheral Identification Register 0

The PMPIDR0 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

PMPIDR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|--------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PART_0 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, least significant byte.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMPIDR0

PMPIDR0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xFE0 | PMPIDR0 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

PMPIDR1, Performance Monitors Peripheral Identification Register 1

The PMPIDR1 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

PMPIDR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---|--------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | DES_0 | | | | PART_1 | | | |

Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, least significant nibble of JEP106 ID code. For Arm Limited, this field is 0b1011.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PART_1, bits [3:0]

Part number, most significant nibble.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMPIDR1

PMPIDR1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xFE4 | PMPIDR1 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMPIDR2, Performance Monitors Peripheral Identification Register 2

The PMPIDR2 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

PMPIDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|-------|----|-------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | REVISION | | | | JEDEC | | DES_1 | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Part major revision. Parts can also use this field to extend Part number to 16-bits.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

Indicates a JEP106 identity code is used.

Reads as 0b1.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, most significant bits of JEP106 ID code. For Arm Limited, this field is 0b011.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMPIDR2

PMPIDR2 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0xFE8 | PMPIDR2 |

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMPIDR3, Performance Monitors Peripheral Identification Register 3

The PMPIDR3 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

PMPIDR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|--------|---|---|---|------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | REVAND | | | | CMOD | | | |

Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Part minor revision. Parts using [PMPIDR2](#).REVISION as an extension to the Part number must use this field as a major revision number.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer modified. Indicates someone other than the Designer has modified the component.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMPIDR3

PMPIDR3 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
|-----------|--------|----------|

| | | |
|-----|-------|---------|
| PMU | 0xFEC | PMPIDR3 |
|-----|-------|---------|

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMPIDR4, Performance Monitors Peripheral Identification Register 4

The PMPIDR4 characteristics are:

Purpose

Provides information to identify a Performance Monitor component.

For more information, see 'About the Peripheral identification scheme'.

Configuration

Implementation of this register is OPTIONAL.

If FEAT_DoPD is implemented, this register is in the Core power domain. If FEAT_DoPD is not implemented, this register is in the Debug power domain.

This register is required for CoreSight compliance.

Attributes

PMPIDR4 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|------|---|---|---|-------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | SIZE | | | | DES_2 | | | |

Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

Size of the component. Log₂ of the number of 4KB pages from the start of the component to the end of the component ID registers.

Reads as 0b0000.

Access to this field is **RO**.

DES_2, bits [3:0]

Designer, JEP106 continuation code, least significant nibble. For Arm Limited, this field is 0b0100.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing PMPIDR4

PMPIDR4 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
|-----------|--------|----------|

| | | |
|-----|-------|---------|
| PMU | 0xFD0 | PMPIDR4 |
|-----|-------|---------|

This interface is accessible as follows:

- When FEAT_DoPD is not implemented or IsCorePowered() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMSWINC_EL0, Performance Monitors Software Increment register

The PMSWINC_EL0 characteristics are:

Purpose

Increments a counter that is configured to count the Software increment event, event 0x00. For more information, see 'SW_INCR'.

Configuration

External register PMSWINC_EL0 bits [31:0] are architecturally mapped to AArch64 System register [PMSWINC_EL0\[31:0\]](#).

External register PMSWINC_EL0 bits [31:0] are architecturally mapped to AArch32 System register [PMSWINC\[31:0\]](#).

PMSWINC_EL0 is in the Core power domain.

Implementation of this register is OPTIONAL.

If this register is implemented, use of it is deprecated.

If 1 is written to bit [n] from the external debug interface, it is CONSTRAINED UNPREDICTABLE whether or not a SW_INCR event is created for counter n. This is consistent with not implementing the register in the external debug interface.

Attributes

PMSWINC_EL0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|----|----|----|----|----|----|----|----|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | P30 | P29 | P28 | P27 | P26 | P25 | P24 | P23 | P22 | P21 | P20 | P19 | P18 | P17 | P16 | P15 | P14 | P13 | P12 | P11 | P10 | P9 | P8 | P7 | P6 | P5 | P4 | P3 | P2 | P1 | P0 |

Bit [31]

Reserved, RES0.

P<n>, bit [n], for n = 30 to 0

Event counter software increment bit for [PMEVCNTR<n>_EL0](#).

If [PMCFGR.N](#) is less than 31, bits [30:[PMCFGR.N](#)] are WI.

| P<n> | Meaning |
|------|---|
| 0b0 | No action. The write to this bit is ignored. |
| 0b1 | It is CONSTRAINED UNPREDICTABLE whether a SW_INCR event is generated for event counter n. |

Accessing PMSWINC_EL0

Note

SoftwareLockStatus() depends on the type of access attempted and AllowExternalPMUAccess() has a new definition from Armv8.4. Refer to the Pseudocode definitions for more information.

PMSWINC_EL0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-------------|
| PMU | 0xCA0 | PMSWINC_EL0 |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and SoftwareLockStatus() accesses to this register are **WI**.
- When IsCorePowered(), !DoubleLockStatus(), !OSLockStatus(), AllowExternalPMUAccess() and !SoftwareLockStatus() accesses to this register are **WO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

PMVIDSR, VMID Sample Register

The PMVIDSR characteristics are:

Purpose

Contains the sampled VMID value that is captured on reading [PMPCSR](#)[31:0].

Configuration

PMVIDSR is in the Core power domain.

This register is present only when FEAT_PCSRv8p2 is implemented and EL2 is implemented. Otherwise, direct accesses to PMVIDSR are RES0.

Note

Before Armv8.2, the PC Sample-based Profiling Extension can be implemented in the external debug register space, as indicated by the value of [EDDEVID](#).PCSample.

Attributes

PMVIDSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|----|----|---|---|------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | VMID[15:8] | | | | | | | | VMID | | | | | | | |

Bits [31:16]

Reserved, RES0.

VMID[15:8], bits [15:8]

When FEAT_VMID16 is implemented:

Extension to VMID[7:0]. For more information, see VMID[7:0].

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

VMID, bits [7:0]

VMID sample. The VMID associated with the most recent [PMPCSR](#) sample. When the most recent [PMPCSR](#) sample was generated:

- This field is set to an UNKNOWN value if any of the following apply:
 - EL2 is disabled in the current Security state.
 - The PE is executing at EL2.

- EL2 is enabled in the current Security state, the PE is executing at EL0, EL2 is using AArch64, HCR_EL2.E2H == 1, and HCR_EL2.TGE == 1.
- Otherwise:
 - If EL2 is using AArch64 and either FEAT_VMID16 is not implemented or [VTCR_EL2.VS](#) is 1, this field is set to [VTTBR_EL2.VMID](#).
 - If EL2 is using AArch64, FEAT_VMID16 is implemented, and [VTCR_EL2.VS](#) is 0, PMVIDSR.VMID[7:0] is set to [VTTBR_EL2.VMID\[7:0\]](#) and PMVIDSR.VMID[15:8] is RES0.
 - If EL2 is using AArch32, this field is set to [VTTBR.VMID](#).

Because the value written to PMVIDR is an indirect read of the VMID value, it is CONSTRAINED UNPREDICTABLE whether PMVIDSR is set to the original or new value if [PMPCSR](#) samples:

- An instruction that writes to the VMID value.
- The next Context synchronization event.
- Any instruction executed between these two instructions.

The reset behavior of this field is:

- On a Cold reset, this field resets to an architecturally UNKNOWN value.

Accessing PMVIDSR

IMPLEMENTATION DEFINED extensions to external debug might make the value of this register UNKNOWN, see 'Permitted behavior that might make the PC Sample-based profiling registers UNKNOWN'.

PMVIDSR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| PMU | 0x20C | PMVIDSR |

This interface is accessible as follows:

- When IsCorePowered(), !DoubleLockStatus() and !OSLockStatus() accesses to this register are **RO**.
- Otherwise accesses to this register generate an error response.

30/09/2021 14:52; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCACATR<n>, Address Comparator Access Type Register <n>, n = 0 - 15

The TRCACATR<n> characteristics are:

Purpose

Defines the type of access for the corresponding [TRCACVR<n>](#) Register. This register configures the context type, Exception levels, alignment, masking that is applied by the Address Comparator, and how the Address Comparator behaves when it is one half of an Address Range Comparator.

Configuration

External register TRCACATR<n> bits [63:0] are architecturally mapped to AArch64 System register [TRCACATR<n>\[63:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMACPAIRS * 2 > n. Otherwise, direct accesses to TRCACATR<n> are RES0.

Attributes

TRCACATR<n> is a 64-bit register.

Field descriptions

| | | | | | | |
|----------------------------|----|----|----|----|----|----|
| 63626160595857565554535251 | 50 | 49 | 48 | 47 | 46 | 45 |
| | | | | | | |
| | | | | | | |
| 31302928272625242322212019 | 18 | 17 | 16 | 15 | 14 | 13 |

Bits [63:19]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [18]

When TRCIDR6.EXLEVEL_RL_EL2 == 1:

Realm EL2 address comparison control. Controls whether a comparison can occur at EL2 in Realm state.

| EXLEVEL_RL_EL2 | Meaning |
|----------------|--|
| 0b0 | When TRCACATR<n>.EXLEVEL_NS_EL2 is 0 the Address Comparator performs comparisons in Realm EL2. When TRCACATR<n>.EXLEVEL_NS_EL2 is 1 the Address Comparator does not perform comparisons in Realm EL2. |
| 0b1 | When TRCACATR<n>.EXLEVEL_NS_EL2 is 0 the Address Comparator does not perform comparisons in Realm EL2. When TRCACATR<n>.EXLEVEL_NS_EL2 is 1 the Address Comparator performs comparisons in Realm EL2. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL1, bit [17]**When TRCIDR6.EXLEVEL_RL_EL1 == 1:**

Realm EL1 address comparison control. Controls whether a comparison can occur at EL1 in Realm state.

| EXLEVEL_RL_EL1 | Meaning |
|----------------|--|
| 0b0 | When TRCACATR<n>.EXLEVEL_NS_EL1 is 0 the Address Comparator performs comparisons in Realm EL1. When TRCACATR<n>.EXLEVEL_NS_EL1 is 1 the Address Comparator does not perform comparisons in Realm EL1. |
| 0b1 | When TRCACATR<n>.EXLEVEL_NS_EL1 is 0 the Address Comparator does not perform comparisons in Realm EL1. When TRCACATR<n>.EXLEVEL_NS_EL1 is 1 the Address Comparator performs comparisons in Realm EL1. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL0, bit [16]**When TRCIDR6.EXLEVEL_RL_EL0 == 1:**

Realm EL0 address comparison control. Controls whether a comparison can occur at EL0 in Realm state.

| EXLEVEL_RL_EL0 | Meaning |
|----------------|--|
| 0b0 | When TRCACATR<n>.EXLEVEL_NS_EL0 is 0 the Address Comparator performs comparisons in Realm EL0. When TRCACATR<n>.EXLEVEL_NS_EL0 is 1 the Address Comparator does not perform comparisons in Realm EL0. |
| 0b1 | When TRCACATR<n>.EXLEVEL_NS_EL0 is 0 the Address Comparator does not perform comparisons in Realm EL0. When TRCACATR<n>.EXLEVEL_NS_EL0 is 1 the Address Comparator performs comparisons in Realm EL0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [15]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [14]**When Non-secure EL2 is implemented:**

Non-secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Non-secure state.

| EXLEVEL_NS_EL2 | Meaning |
|----------------|--|
| 0b0 | The Address Comparator performs comparisons in Non-secure EL2. |
| 0b1 | The Address Comparator does not perform comparisons in Non-secure EL2. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL1, bit [13]**When Non-secure EL1 is implemented:**

Non-secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Non-secure state.

| EXLEVEL_NS_EL1 | Meaning |
|----------------|--|
| 0b0 | The Address Comparator performs comparisons in Non-secure EL1. |
| 0b1 | The Address Comparator does not perform comparisons in Non-secure EL1. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL0, bit [12]**When Non-secure EL0 is implemented:**

Non-secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Non-secure state.

| EXLEVEL_NS_EL0 | Meaning |
|----------------|--|
| 0b0 | The Address Comparator performs comparisons in Non-secure EL0. |
| 0b1 | The Address Comparator does not perform comparisons in Non-secure EL0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL3, bit [11]**When EL3 is implemented:**

EL3 address comparison control. Controls whether a comparison can occur at EL3.

| EXLEVEL_S_EL3 | Meaning |
|----------------------|---|
| 0b0 | The Address Comparator performs comparisons in EL3. |
| 0b1 | The Address Comparator does not perform comparisons in EL3. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL2, bit [10]**When EL2 is implemented and FEAT_SEL2 is implemented:**

Secure EL2 address comparison control. Controls whether a comparison can occur at EL2 in Secure state.

| EXLEVEL_S_EL2 | Meaning |
|----------------------|--|
| 0b0 | The Address Comparator performs comparisons in Secure EL2. |
| 0b1 | The Address Comparator does not perform comparisons in Secure EL2. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL1, bit [9]**When Secure EL1 is implemented:**

Secure EL1 address comparison control. Controls whether a comparison can occur at EL1 in Secure state.

| EXLEVEL_S_EL1 | Meaning |
|----------------------|--|
| 0b0 | The Address Comparator performs comparisons in Secure EL1. |
| 0b1 | The Address Comparator does not perform comparisons in Secure EL1. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL0, bit [8]**When Secure EL0 is implemented:**

Secure EL0 address comparison control. Controls whether a comparison can occur at EL0 in Secure state.

| EXLEVEL S EL0 | Meaning |
|---------------|--|
| 0b0 | The Address Comparator performs comparisons in Secure EL0. |
| 0b1 | The Address Comparator does not perform comparisons in Secure EL0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [7]

Reserved, RES0.

CONTEXT, bits [6:4]

When TRCIDR4.NUMCIDC != 0b0000 or TRCIDR4.NUMVMIDC != 0b0000:

Selects a Context Identifier Comparator or Virtual Context Identifier Comparator:

| CONTEXT | Meaning | Applies when |
|---------|---------------|--|
| 0b000 | Comparator 0. | |
| 0b001 | Comparator 1. | When TRCIDR4.NUMCIDC > 0b0001 or TRCIDR4.NUMVMIDC > 0b0001 |
| 0b010 | Comparator 2. | When TRCIDR4.NUMCIDC > 0b0010 or TRCIDR4.NUMVMIDC > 0b0010 |
| 0b011 | Comparator 3. | When TRCIDR4.NUMCIDC > 0b0011 or TRCIDR4.NUMVMIDC > 0b0011 |
| 0b100 | Comparator 4. | When TRCIDR4.NUMCIDC > 0b0100 or TRCIDR4.NUMVMIDC > 0b0100 |
| 0b101 | Comparator 5. | When TRCIDR4.NUMCIDC > 0b0101 or TRCIDR4.NUMVMIDC > 0b0101 |
| 0b110 | Comparator 6. | When TRCIDR4.NUMCIDC > 0b0110 or TRCIDR4.NUMVMIDC > 0b0110 |
| 0b111 | Comparator 7. | When TRCIDR4.NUMCIDC > 0b0111 or TRCIDR4.NUMVMIDC > 0b0111 |

The width of this field is dependent on the maximum number of Context Identifier Comparators or Virtual Context Identifier Comparators implemented. Unimplemented bits are RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CONTEXTTYPE, bits [3:2]

When TRCIDR4.NUMCIDC != 0b0000 or TRCIDR4.NUMVMIDC != 0b0000:

Controls whether the Address Comparator is dependent on a Context Identifier Comparator, a Virtual Context Identifier Comparator, or both comparisons.

| CONTEXTTYPE | Meaning | Applies when |
|-------------|---|---|
| 0b00 | The Address Comparator is not dependent on the Context Identifier Comparators or Virtual Context Identifier Comparators. | |
| 0b01 | The Address Comparator is dependent on the Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if both the Context Identifier Comparator and the address comparison match. | When TRCIDR4.NUMCIDC != 0b0000 |
| 0b10 | The Address Comparator is dependent on the Virtual Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if both the Virtual Context Identifier Comparator and the address comparison match. | When TRCIDR4.NUMVMIDC != 0b0000 |
| 0b11 | The Address Comparator is dependent on the Context Identifier Comparator and Virtual Context Identifier Comparator that TRCACATR<n>.CONTEXT specifies. The Address Comparator signals a match only if the Context Identifier Comparator, the Virtual Context Identifier Comparator, and address comparison all match. | When TRCIDR4.NUMCIDC != 0b0000 and TRCIDR4.NUMVMIDC != 0b0000 |

If [TRCIDR4.NUMCIDC](#) == 0b0000, then bit [2] is RES0.

If [TRCIDR4.NUMVMIDC](#) == 0b0000, then bit [3] is RES0.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [1:0]

Reserved, RES0.

Accessing TRCACATR<n>

Must be programmed if any of the following are true:

- [TRCBBCTL](#)R.RANGE[n/2] == 1.
- [TRCRSCTL](#)R<a>.GROUP == 0b0100 and [TRCRSCTL](#)R<a>.SAC[n] == 1.
- [TRCRSCTL](#)R<a>.GROUP == 0b0101 and [TRCRSCTL](#)R<a>.ARC[n/2] == 1.
- [TRCVIIECTL](#)R.EXCLUDE[n/2] == 1.
- [TRCVIIECTL](#)R.INCLUDE[n/2] == 1.
- [TRCVISSCTL](#)R.START[n] == 1.
- [TRCVISSCTL](#)R.STOP[n] == 1.
- TRCSSCCR<>.ARC[n/2] == 1.
- TRCSSCCR<>.SAC[n] == 1.
- [TRCQCTL](#)R.RANGE[n/2] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCACATR<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|-------------|
| ETE | 0x480 + (8 * n) | TRCACATR<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCACVR<n>, Address Comparator Value Register <n>, n = 0 - 15

The TRCACVR<n> characteristics are:

Purpose

Contains the address value.

Configuration

External register TRCACVR<n> bits [63:0] are architecturally mapped to AArch64 System register [TRCACVR<n>\[63:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMACPAIRS * 2 > n. Otherwise, direct accesses to TRCACVR<n> are RES0.

Attributes

TRCACVR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| ADDRESS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| ADDRESS | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

ADDRESS, bits [63:0]

Address Value.

The Address Comparators can support implementations that use multiple address widths. When the trace unit compares the ADDRESS field with an address that has a width less than this field, then the address must be zero-extended to the ADDRESS field width. The trace unit then compares all implemented bits. For example, in a system that supports both 32-bit and 64-bit addresses, when the PE is in AArch32 state the comparator must zero-extend the 32-bit address and compare against the full 64 bits that are stored in the TRCACVR<n>. This requires that the trace analyzer always programs all implemented bits of the TRCACVR<n>.

The result of writing a value other than all zeros or all ones to ADDRESS at bits[63:P] is an UNKNOWN value, where P is defined as the maximum virtual address size supported by the PE.

The result of writing a value of all zeros or all ones to ADDRESS at bits[63:P] is the written value, and a read of the register returns the written value.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCACVR<n>

Must be programmed if any of the following are true:

- [TRCBBCTLR](#).RANGE[n/2] == 1.
- [TRCRSCTLR<a>](#).GROUP == 0b0100 and [TRCRSCTLR<a>](#).SAC[n] == 1.
- [TRCRSCTLR<a>](#).GROUP == 0b0101 and [TRCRSCTLR<a>](#).ARC[n/2] == 1.
- [TRCVIIECTLR](#).EXCLUDE[n/2] == 1.
- [TRCVIIECTLR](#).INCLUDE[n/2] == 1.

- [TRCVISSCTLR](#).START[n] == 1.
- [TRCVISSCTLR](#).STOP[n] == 1.
- TRCSSCCR<>.ARC[n/2] == 1.
- TRCSSCCR<>.SAC[n] == 1.
- [TRCQCTLR](#).RANGE[n/2] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCACVR<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|------------|
| ETE | 0x400 + (8 * n) | TRCACVR<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCAUTHSTATUS, Authentication Status Register

The TRCAUTHSTATUS characteristics are:

Purpose

Provides information about the state of the IMPLEMENTATION DEFINED authentication interface for debug.

For additional information, see the CoreSight Architecture Specification.

Configuration

External register TRCAUTHSTATUS bits [31:0] are architecturally mapped to AArch64 System register [TRCAUTHSTATUS\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCAUTHSTATUS are RES0.

Attributes

TRCAUTHSTATUS is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

Bits [31:28]

Reserved, RES0.

RTNID, bits [27:26]

Root non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RTNID.

RTID, bits [25:24]

Root invasive debug.

| RTID | Meaning |
|------|------------------|
| 0b00 | Not implemented. |

Bits [23:16]

Reserved, RES0.

RLNID, bits [15:14]

Realm non-invasive debug.

This field has the same value as DBGAUTHSTATUS_EL1.RLNID.

RLID, bits [13:12]

Realm invasive debug.

| RLID | Meaning |
|------|------------------|
| 0b00 | Not implemented. |

HNID, bits [11:10]

Hyp Non-invasive Debug. Indicates whether a separate enable control for EL2 non-invasive debug features is implemented and enabled.

| HNID | Meaning |
|------|---|
| 0b00 | Separate Hyp non-invasive debug enable not implemented, or EL2 non-invasive debug features not implemented. |
| 0b10 | Implemented and disabled. |
| 0b11 | Implemented and enabled. |

All other values are reserved.

This field reads as 0b00.

HID, bits [9:8]

Hyp Invasive Debug. Indicates whether a separate enable control for EL2 invasive debug features is implemented and enabled.

| HID | Meaning |
|------|---|
| 0b00 | Separate Hyp invasive debug enable not implemented, or EL2 invasive debug features not implemented. |
| 0b10 | Implemented and disabled. |
| 0b11 | Implemented and enabled. |

All other values are reserved.

This field reads as 0b00.

SNID, bits [7:6]

Secure Non-invasive Debug. Indicates whether Secure non-invasive debug features are implemented and enabled.

| SNID | Meaning |
|------|---|
| 0b00 | Secure non-invasive debug features not implemented. |
| 0b10 | Implemented and disabled. |
| 0b11 | Implemented and enabled. |

All other values are reserved.

When EL3 is implemented, this field takes the value 0b10 or 0b11 depending whether Secure non-invasive debug is enabled.

When EL3 is not implemented and the PE is Non-secure, this field reads as 0b00.

When EL3 is not implemented and the PE is Secure, this field takes the value 0b10 or 0b11 depending whether Secure non-invasive debug is enabled.

SID, bits [5:4]

Secure Invasive Debug. Indicates whether Secure invasive debug features are implemented and enabled.

| SID | Meaning |
|------|---|
| 0b00 | Secure invasive debug features not implemented. |
| 0b10 | Implemented and disabled. |
| 0b11 | Implemented and enabled. |

All other values are reserved.

This field reads as 0b00.

NSNID, bits [3:2]

Non-secure Non-invasive Debug. Indicates whether Non-secure non-invasive debug features are implemented and enabled.

| NSNID | Meaning |
|-------|---|
| 0b00 | Non-secure non-invasive debug features not implemented. |
| 0b10 | Implemented and disabled. |
| 0b11 | Implemented and enabled. |

All other values are reserved.

When EL3 is implemented, this field reads as 0b11.

When EL3 is not implemented and the PE is Non-secure, this field reads as 0b11.

When EL3 is not implemented and the PE is Secure, this field reads as 0b00.

NSID, bits [1:0]

Non-secure Invasive Debug. Indicates whether Non-secure invasive debug features are implemented and enabled.

| NSID | Meaning |
|------|---|
| 0b00 | Non-secure invasive debug features not implemented. |
| 0b10 | Implemented and disabled. |
| 0b11 | Implemented and enabled. |

All other values are reserved.

This field reads as 0b00.

Accessing TRCAUTHSTATUS

For implementations that support multiple access mechanisms, different access mechanisms can return different values for reads of TRCAUTHSTATUS if the authentication signals have changed and that change has not yet been synchronized by a Context synchronization event. This scenario can happen if, for example, the external debugger view is implemented separately from the system instruction view to allow for separate power domains, and so observes changes on the signals differently.

External debugger accesses to this register are unaffected by the OS Lock.

TRCAUTHSTATUS can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|---------------|
| ETE | 0xFB8 | TRCAUTHSTATUS |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCAUXCTLR, Auxiliary Control Register

The TRCAUXCTLR characteristics are:

Purpose

The function of this register is IMPLEMENTATION DEFINED.

Configuration

External register TRCAUXCTLR bits [31:0] are architecturally mapped to AArch64 System register [TRCAUXCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCAUXCTLR are RES0.

Attributes

TRCAUXCTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to 0.

Accessing TRCAUXCTLR

If this register is nonzero then it might cause the behavior of a trace unit to contradict this architecture specification. See the documentation of the specific implementation for information about the IMPLEMENTATION DEFINED support for this register.

TRCAUXCTLR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|------------|
| ETE | 0x018 | TRCAUXCTLR |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

TRCBBCTLR, Branch Broadcast Control Register

The TRCBBCTLR characteristics are:

Purpose

Controls the regions in the memory map where branch broadcasting is active.

Configuration

External register TRCBBCTLR bits [31:0] are architecturally mapped to AArch64 System register [TRCBBCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented, TRCIDR0.TRCBB == 1 and TRCIDR4.NUMACPAIRS > 0b0000. Otherwise, direct accesses to TRCBBCTLR are RES0.

Attributes

TRCBBCTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|------|----|----------|----|----------|----|----------|----|----------|----|----------|----|----------|---|----------|---|----------|---|----------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | MODE | | RANGE[7] | | RANGE[6] | | RANGE[5] | | RANGE[4] | | RANGE[3] | | RANGE[2] | | RANGE[1] | | RANGE[0] | | RANGE[0] | | | |

Bits [31:9]

Reserved, RES0.

MODE, bit [8]

Mode.

| MODE | Meaning |
|------|---|
| 0b0 | Exclude Mode. Branch broadcasting is not active for instructions in the address ranges defined by TRCBBCTLR.RANGE. If TRCBBCTLR.RANGE == 0x00 then branch broadcasting is active for all instructions. |
| 0b1 | Include Mode. Branch broadcasting is active for instructions in the address ranges defined by TRCBBCTLR.RANGE. If TRCBBCTLR.RANGE == 0x00 then the behavior of the trace unit is CONSTRAINED UNPREDICTABLE. That is, the trace unit might or might not consider any instructions to be in a branch broadcasting region. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RANGE[<m>], bit [m], for m = 7 to 0

Address range field.

Selects whether Address Range Comparator <m> is used with branch broadcasting.

| RANGE[<m>] | Meaning |
|-------------------------|---|
| 0b0 | The address range that Address Range Comparator <m> defines, is not selected. |
| 0b1 | The address range that Address Range Comparator <m> defines, is selected. |

This bit is RES0 if m >= [TRCIDR4](#).NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCBBCTL

Must be programmed if [TRCCONFIGR](#).BB == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCBBCTL can be accessed through the external debug interface:

| Component | Offset | Instance |
|------------------|---------------|-----------------|
| ETE | 0x03C | TRCBBCTL |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCCCTLR, Cycle Count Control Register

The TRCCCCTLR characteristics are:

Purpose

Set the threshold value for cycle counting.

Configuration

External register TRCCCCTLR bits [31:0] are architecturally mapped to AArch64 System register [TRCCCCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR0.TRCCCI == 1. Otherwise, direct accesses to TRCCCCTLR are RES0.

Attributes

TRCCCCTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----------|----|---|---|---|---|---|---|---|---|---|---|--|--|--|--|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | | | | |
| RES0 | | | | | | | | | | | | | | | | | | | | THRESHOLD | | | | | | | | | | | | | | | | |

Bits [31:12]

Reserved, RES0.

THRESHOLD, bits [11:0]

Sets the threshold value for instruction trace cycle counting.

The minimum threshold value that can be programmed into THRESHOLD is given in [TRCIDR3.CCITMIN](#). If the THRESHOLD value is smaller than the value in [TRCIDR3.CCITMIN](#) then the behavior is CONSTRAINED UNPREDICTABLE. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

Writing a value of zero when [TRCONFIGR.CCI](#) enables instruction trace cycle counting results in CONSTRAINED UNPREDICTABLE behavior. That is, cycle counts might or might not be included in the trace and the cycle count threshold is not known.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCCCTLR

Must be programmed if [TRCONFIGR.CCI](#) == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCCCTLR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| ETE | 0x038 | TRCCCCTLR |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDCCTL0, Context Identifier Comparator Control Register 0

The TRCCIDCCTL0 characteristics are:

Purpose

Contains Context identifier mask values for the [TRCCIDCVR<n>](#) registers, for n = 0 to 3.

Configuration

External register TRCCIDCCTL0 bits [31:0] are architecturally mapped to AArch64 System register [TRCCIDCCTL0\[31:0\]](#).

This register is present only when FEAT_ETE is implemented, TRCIDR4.NUMCIDC > 0x0 and TRCIDR2.CIDSIZE > 0b00000. Otherwise, direct accesses to TRCCIDCCTL0 are RES0.

Attributes

TRCCIDCCTL0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | |
| COMP3[7] | COMP3[6] | COMP3[5] | COMP3[4] | COMP3[3] | COMP3[2] | COMP3[1] | COMP3[0] | COMP2[7] | COMP2[6] | COMP2[5] | COMP2[4] |

COMP3[<m>], bit [m+24], for m = 7 to 0
When TRCIDR4.NUMCIDC > 3:

TRCCIDCVR3 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR3. Each bit in this field corresponds to a byte in TRCCIDCVR3.

| COMP3[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR3[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR3[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP2[<m>], bit [m+16], for m = 7 to 0
When TRCIDR4.NUMCIDC > 2:

TRCCIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR2. Each bit in this field corresponds to a byte in TRCCIDCVR2.

| COMP2[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR2[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR2[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP1[<m>], bit [m+8], for m = 7 to 0

When TRCIDR4.NUMCIDC > 1:

TRCCIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR1. Each bit in this field corresponds to a byte in TRCCIDCVR1.

| COMP1[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR1[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR1[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP0[<m>], bit [m], for m = 7 to 0

When TRCIDR4.NUMCIDC > 0:

TRCCIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR0. Each bit in this field corresponds to a byte in TRCCIDCVR0.

| COMP0[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR0[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR0[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCCIDCCTL0

If software uses the [TRCCIDCVR<n>](#) registers, for n = 0 to 3, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in [TRCCIDCVR<n>](#) to 0x00.

If any bit is 1 and the relevant byte in [TRCCIDCVR<n>](#) is not 0x00, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCIDCCTL0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-------------|
| ETE | 0x680 | TRCCIDCCTL0 |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDCCTL1, Context Identifier Comparator Control Register 1

The TRCCIDCCTL1 characteristics are:

Purpose

Contains Context identifier mask values for the [TRCCIDCVR<n>](#) registers, for n = 4 to 7.

Configuration

External register TRCCIDCCTL1 bits [31:0] are architecturally mapped to AArch64 System register [TRCCIDCCTL1\[31:0\]](#).

This register is present only when FEAT_ETE is implemented, TRCIDR4.NUMCIDC > 0x4 and TRCIDR2.CIDSIZE > 0b00000. Otherwise, direct accesses to TRCCIDCCTL1 are RES0.

Attributes

TRCCIDCCTL1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | |
| COMP7[7] | COMP7[6] | COMP7[5] | COMP7[4] | COMP7[3] | COMP7[2] | COMP7[1] | COMP7[0] | COMP6[7] | COMP6[6] | COMP6[5] | COMP6[4] |

COMP7[<m>], bit [m+24], for m = 7 to 0
When TRCIDR4.NUMCIDC > 7:

TRCCIDCVR7 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR7. Each bit in this field corresponds to a byte in TRCCIDCVR7.

| COMP7[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR7[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR7[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP6[<m>], bit [m+16], for m = 7 to 0
When TRCIDR4.NUMCIDC > 6:

TRCCIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR6. Each bit in this field corresponds to a byte in TRCCIDCVR6.

| COMP6[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR6[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP5[<m>], bit [m+8], for m = 7 to 0

When TRCIDR4.NUMCICD > 5:

TRCCIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR5. Each bit in this field corresponds to a byte in TRCCIDCVR5.

| COMP5[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR5[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP4[<m>], bit [m], for m = 7 to 0

When TRCIDR4.NUMCICD > 4:

TRCCIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCCIDCVR4. Each bit in this field corresponds to a byte in TRCCIDCVR4.

| COMP4[<m>] | Meaning |
|------------|---|
| 0b0 | The trace unit includes TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison. |
| 0b1 | The trace unit ignores TRCCIDCVR4[(m×8+7):(m×8)] when it performs the Context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.CIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCCIDCCTLR1

If software uses the [TRCCIDCVR<n>](#) registers, for n = 4 to 7, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in [TRCCIDCVR<n>](#) to 0x00.

If any bit is 1 and the relevant byte in [TRCCIDCVR<n>](#) is not 0x00, the behavior of the Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCIDCCTLR1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|--------------|
| ETE | 0x684 | TRCCIDCCTLR1 |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDCVR<n>, Context Identifier Comparator Value Registers <n>, n = 0 - 7

The TRCCIDCVR<n> characteristics are:

Purpose

Contains a Context identifier value.

Configuration

External register TRCCIDCVR<n> bits [63:0] are architecturally mapped to AArch64 System register [TRCCIDCVR<n>\[63:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMCIDC > n. Otherwise, direct accesses to TRCCIDCVR<n> are RES0.

Attributes

TRCCIDCVR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

VALUE, bits [63:0]

Context identifier value. The width of this field is indicated by [TRCIDR2.CIDSIZE](#). Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Context identifier is zero until the PE updates the Context identifier.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCIDCVR<n>

Must be programmed if any of the following are true:

- [TRCRSCTLR<a>.GROUP](#) == 0b0110 and [TRCRSCTLR<a>.CID\[n\]](#) == 1.
- [TRCACATR<a>.CONTEXTTYPE](#) == 0b01 or 0b11 and [TRCACATR<a>.CONTEXT](#) == n.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCIDCVR<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|--------------|
| ETE | 0x600 + (8 * n) | TRCCIDCVR<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.

- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDR0, Component Identification Register 0

The TRCCIDR0 characteristics are:

Purpose

Provides discovery information about the component.
For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCCIDR0 are RES0.

Attributes

TRCCIDR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL_0 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_0, bits [7:0]

Component identification preamble, segment 0.
Reads as 0x0D.
Access to this field is **RO**.

Accessing TRCCIDR0

External debugger accesses to this register are unaffected by the OS Lock.

TRCCIDR0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFF0 | TRCCIDR0 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCCIDR1, Component Identification Register 1

The TRCCIDR1 characteristics are:

Purpose

Provides discovery information about the component.
For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCCIDR1 are RES0.

Attributes

TRCCIDR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|---------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | CLASS | | | PRMBL_1 | | | | |

Bits [31:8]

Reserved, RES0.

CLASS, bits [7:4]

Component class.

| CLASS | Meaning |
|--------|-----------------------|
| 0b1001 | CoreSight peripheral. |

Other values are defined by the CoreSight Architecture.

This field reads as 0x9.

PRMBL_1, bits [3:0]

Component identification preamble, segment 1.

Reads as 0b0000.

Access to this field is **RO**.

Accessing TRCCIDR1

External debugger accesses to this register are unaffected by the OS Lock.

TRCCIDR1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFF4 | TRCCIDR1 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCIDR2, Component Identification Register 2

The TRCCIDR2 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCCIDR2 are RES0.

Attributes

TRCCIDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL_2 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_2, bits [7:0]

Component identification preamble, segment 2.

Reads as 0x05.

Access to this field is **RO**.

Accessing TRCCIDR2

External debugger accesses to this register are unaffected by the OS Lock.

TRCCIDR2 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFF8 | TRCCIDR2 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCCIDR3, Component Identification Register 3

The TRCCIDR3 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCCIDR3 are RES0.

Attributes

TRCCIDR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PRMBL_3 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PRMBL_3, bits [7:0]

Component identification preamble, segment 3.

Reads as 0xB1.

Access to this field is **RO**.

Accessing TRCCIDR3

External debugger accesses to this register are unaffected by the OS Lock.

TRCCIDR3 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFFC | TRCCIDR3 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCCLAIMCLR, Claim Tag Clear Register

The TRCCLAIMCLR characteristics are:

Purpose

In conjunction with [TRCCLAIMSET](#), provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information, see the CoreSight Architecture Specification.

Configuration

External register TRCCLAIMCLR bits [31:0] are architecturally mapped to AArch64 System register [TRCCLAIMCLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCCLAIMCLR are RES0.

Attributes

TRCCLAIMCLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | |
|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 |
| CLR[31] | CLR[30] | CLR[29] | CLR[28] | CLR[27] | CLR[26] | CLR[25] | CLR[24] | CLR[23] | CLR[22] | CLR[21] | CLR[20] | CLR[19] | CLR[18] |

CLR[<m>], bit [m], for m = 31 to 0

Claim Tag Clear. Indicates the current status of Claim Tag bit <m>, and is used to clear Claim Tag bit <m> to 0.

| CLR[<m>] | Meaning |
|----------|---|
| 0b0 | On a read: Claim Tag bit <m> is not set. On a write: Ignored. |
| 0b1 | On a read: Claim Tag bit <m> is set. On a write: Clear Claim tag bit <m> to 0. |

The number of Claim Tag bits implemented is indicated in [TRCCLAIMSET](#).

This bit reads-as-zero and ignores writes if m > the number of Claim Tag bits.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to 0.

Access to this field is **W1C**.

Accessing TRCCLAIMCLR

TRCCLAIMCLR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-------------|
| ETE | 0xFA4 | TRCCLAIMCLR |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

TRCCLAIMSET, Claim Tag Set Register

The TRCCLAIMSET characteristics are:

Purpose

In conjunction with [TRCCLAIMCLR](#), provides Claim Tag bits that can be separately set and cleared to indicate whether functionality is in use by a debug agent.

For additional information, see the CoreSight Architecture Specification.

Configuration

External register TRCCLAIMSET bits [31:0] are architecturally mapped to AArch64 System register [TRCCLAIMSET\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCCLAIMSET are RES0.

The number of claim tag bits implemented is IMPLEMENTATION DEFINED. Arm recommends that implementations support a minimum of four claim tag bits, that is, SET[3:0] reads as 0b1111.

Attributes

TRCCLAIMSET is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|---------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SET[31] | SET[30] | SET[29] | SET[28] | SET[27] | SET[26] | SET[25] | SET[24] | SET[23] | SET[22] | SET[21] | SET[20] | SET[19] | SET[18] | SET[17] | SET[16] | SET[15] | SET[14] | SET[13] | SET[12] | SET[11] | SET[10] | SET[9] | SET[8] | SET[7] | SET[6] | SET[5] | SET[4] | SET[3] | SET[2] | SET[1] | SET[0] |

SET[<m>], bit [m], for m = 31 to 0

Claim Tag Set. Indicates whether Claim Tag bit <m> is implemented, and is used to set Claim Tag bit <m> to 1.

| SET[<m>] | Meaning |
|----------|---|
| 0b0 | On a read: Claim Tag bit <m> is not implemented. On a write: Ignored. |
| 0b1 | On a read: Claim Tag bit <m> is implemented. On a write: Set Claim Tag bit <m> to 1. |

This bit reads-as-zero and ignores writes if m > the number of Claim Tag bits.

Access to this field is **RAO/W1S**.

Accessing TRCCLAIMSET

TRCCLAIMSET can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-------------|
| ETE | 0xFA0 | TRCCLAIMSET |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

TRCCNTCTLR<n>, Counter Control Register <n>, n = 0 - 3

The TRCCNTCTLR<n> characteristics are:

Purpose

Controls the operation of Counter <n>.

Configuration

External register TRCCNTCTLR<n> bits [31:0] are architecturally mapped to AArch64 System register [TRCCNTCTLR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMCNTR > n. Otherwise, direct accesses to TRCCNTCTLR<n> are RES0.

Attributes

TRCCNTCTLR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|---------|---------------|------|--------------|---------------|------|--------------|------|--------------|------|--------------|------|--------------|------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 |
| RES0 | | | | | | | | | | | | | | CNTCHAIN | RLDSELF | RLDEVENT_TYPE | RES0 | RLDEVENT_SEL | CNTEVENT_TYPE | RES0 | CNTEVENT_SEL | RES0 | CNTEVENT_SEL | RES0 | CNTEVENT_SEL | RES0 | CNTEVENT_SEL | RES0 |

Bits [31:18]

Reserved, RES0.

CNTCHAIN, bit [17]

For TRCCNTCTLR3 and TRCCNTCTLR1, this field controls whether the Counter decrements when a reload event occurs for Counter <n-1>.

| CNTCHAIN | Meaning |
|----------|--|
| 0b0 | The Counter does not decrement when a reload event for Counter <n-1> occurs. |
| 0b1 | Counter <n> decrements when a reload event for Counter <n-1> occurs. This concatenates Counter <n> and Counter <n-1>, to provide a larger count value. |

CNTCHAIN is not implemented for TRCCNTCTLR0 and TRCCNTCTLR2.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RLDSELF, bit [16]

Controls whether a reload event occurs for the Counter, when the Counter reaches zero.

| RLDSELF | Meaning |
|---------|--|
| 0b0 | Normal mode. The Counter is in Normal mode. |
| 0b1 | Self-reload mode. The Counter is in Self-reload mode. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RLDEVENT_TYPE, bit [15]

Chooses the type of Resource Selector.

Selects an event, that when it occurs causes a reload event for Counter <n>.

| RLDEVENT_TYPE | Meaning |
|---------------|---|
| 0b0 | A single Resource Selector. TRCCNTCTLR<n>.RLDEVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCCNTCTLR<n>.RLDEVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCCNTCTLR<n>.RLDEVENT.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [14:13]

Reserved, RES0.

RLDEVENT_SEL, bits [12:8]

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTLR<n>.RLDEVENT.TYPE controls whether TRCCNTCTLR<n>.RLDEVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Selects an event, that when it occurs causes a reload event for Counter <n>.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

CNTEVENT_TYPE, bit [7]

Chooses the type of Resource Selector.

Selects an event, that when it occurs causes Counter <n> to decrement.

| CNTEVENT_TYPE | Meaning |
|---------------|---|
| 0b0 | A single Resource Selector. TRCCNTCTLR<n>.CNTEVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCCNTCTLR<n>.CNTEVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCCNTCTLR<n>.CNTEVENT.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

CNTEVENT_SEL, bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCCNTCTLR<n>.CNTEVENT.TYPE controls whether TRCCNTCTLR<n>.CNTEVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Selects an event, that when it occurs causes Counter <n> to decrement.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTCTLR<n>

Must be programmed if [TRCRSCTLR<a>.GROUP == 0b0010](#) and [TRCRSCTLR<a>.COUNTERS\[n\] == 1](#).

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCNTCTLR<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|---------------|
| ETE | 0x150 + (4 * n) | TRCCNTCTLR<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCNTRLDVR<n>, Counter Reload Value Register <n>, n = 0 - 3

The TRCCNTRLDVR<n> characteristics are:

Purpose

This sets or returns the reload count value for Counter <n>.

Configuration

External register TRCCNTRLDVR<n> bits [31:0] are architecturally mapped to AArch64 System register [TRCCNTRLDVR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMCNTR > n. Otherwise, direct accesses to TRCCNTRLDVR<n> are RES0.

Attributes

TRCCNTRLDVR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

VALUE, bits [15:0]

Contains the reload value for Counter <n>. When a reload event occurs for Counter <n> then the trace unit copies the VALUE<n> field into Counter <n>.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTRLDVR<n>

Must be programmed if [TRCRSCTLR<a>](#).GROUP == 0b0010 and [TRCRSCTLR<a>](#).COUNTERS[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCNTRLDVR<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|----------------|
| ETE | 0x140 + (4 * n) | TRCCNTRLDVR<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

TRCCNTVR<n>, Counter Value Register <n>, n = 0 - 3

The TRCCNTVR<n> characteristics are:

Purpose

This sets or returns the value of Counter <n>.

Configuration

External register TRCCNTVR<n> bits [31:0] are architecturally mapped to AArch64 System register [TRCCNTVR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMCNTR > n. Otherwise, direct accesses to TRCCNTVR<n> are RES0.

Attributes

TRCCNTVR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

VALUE, bits [15:0]

Contains the count value of Counter.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCCNTVR<n>

Must be programmed if [TRCRSCTLR<a>](#).GROUP == 0b0010 and [TRCRSCTLR<a>](#).COUNTERS[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCCNTVR<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|-------------|
| ETE | 0x160 + (4 * n) | TRCCNTVR<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

TRCCNTVR<n>, Counter Value Register <n>, n = 0 - 3

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCCONFIGR, Trace Configuration Register

The TRCCONFIGR characteristics are:

Purpose

Controls the tracing options.

Configuration

External register TRCCONFIGR bits [31:0] are architecturally mapped to AArch64 System register [TRCCONFIGR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCCONFIGR are RES0.

Attributes

TRCCONFIGR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------|----|------|------|---------|------|-----|----|------|------|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | VMIDOPT | QE | RSTS | RES0 | VMIDCID | RES0 | CCI | BB | RES0 | RES1 | | | | | | |

Bits [31:16]

Reserved, RES0.

VMIDOPT, bit [15]

When TRCIDR2.VMIDOPT == 0b01:

Virtual context identifier selection control.

| VMIDOPT | Meaning |
|---------|---|
| 0b0 | VTTBR_EL2 .VMID is used as the Virtual context identifier. |
| 0b1 | CONTEXTIDR_EL2 .PROCID is used as the Virtual context identifier. |

When TRCIDR2.VMIDOPT == 0b00:

Reserved, RES0.

Virtual context identifier selection control.

[VTTBR_EL2](#).VMID is used as the Virtual context identifier.

When TRCIDR2.VMIDOPT == 0b10:

Reserved, RES1.

Virtual context identifier selection control.

[CONTEXTIDR_EL2](#).PROCID is used as the Virtual context identifier.

Otherwise:

Reserved, RES0.

QE, bits [14:13]

When TRCIDR0.QSUPP == 0b01:

Q element generation control.

| QE | Meaning |
|------|---|
| 0b00 | Q elements are disabled. |
| 0b01 | Q elements with instruction counts are enabled. |
| | Q elements without instruction counts are disabled. |

All other values are reserved.

When TRCIDR0.QSUPP == 0b10:

Q element generation control.

| QE | Meaning |
|------|--|
| 0b00 | Q elements are disabled. |
| 0b11 | Q elements with instruction counts are enabled. |
| | Q elements without instruction counts are enabled. |

All other values are reserved.

When TRCIDR0.QSUPP == 0b11:

Q element generation control.

| QE | Meaning |
|------|---|
| 0b00 | Q elements are disabled. |
| 0b01 | Q elements with instruction counts are enabled. |
| | Q elements without instruction counts are disabled. |
| 0b11 | Q elements with instruction counts are enabled. |
| | Q elements without instruction counts are enabled. |

All other values are reserved.

Otherwise:

Reserved, RES0.

RS, bit [12]

When TRCIDR0.RETSTACK == 1:

Return stack control.

| RS | Meaning |
|-----|---------------------------|
| 0b0 | Return stack is disabled. |
| 0b1 | Return stack is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TS, bit [11]

When TRCIDR0.TSSIZE != 0b000000:

Global timestamp tracing control.

| TS | Meaning |
|-----|---------------------------------------|
| 0b0 | Global timestamp tracing is disabled. |
| 0b1 | Global timestamp tracing is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [10:8]

Reserved, RES0.

VMID, bit [7]

When TRCIDR2.VMIDSIZE != 0b000000:

Virtual context identifier tracing control.

| VMID | Meaning |
|------|---|
| 0b0 | Virtual context identifier tracing is disabled. |
| 0b1 | Virtual context identifier tracing is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

CID, bit [6]

When TRCIDR2.CIDSIZE != 0b000000:

Context identifier tracing control.

| CID | Meaning |
|-----|---|
| 0b0 | Context identifier tracing is disabled. |
| 0b1 | Context identifier tracing is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [5]

Reserved, RES0.

CCI, bit [4]

When **TRCIDR0.TRCCCI == 1**:

Cycle counting instruction tracing control.

| CCI | Meaning |
|-----|---|
| 0b0 | Cycle counting instruction tracing is disabled. |
| 0b1 | Cycle counting instruction tracing is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

BB, bit [3]

When **TRCIDR0.TRCBB == 1**:

Branch broadcasting control.

| BB | Meaning |
|-----|----------------------------------|
| 0b0 | Branch broadcasting is disabled. |
| 0b1 | Branch broadcasting is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [2:1]

Reserved, RES0.

Bit [0]

Reserved, RES1.

Accessing TRCCONFIGR

Must always be programmed.

TRCCONFIGR.QE must be set to 0b00 if TRCCONFIGR.BB is not 0.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCCONFIGR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|------------|
| ETE | 0x010 | TRCCONFIGR |

This interface is accessible as follows:

- When OSLockStatus(), or !IsTraceCorePowered() or !AllowExternalTraceAccess() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVAFF, Device Affinity Register

The TRCDEVAFF characteristics are:

Purpose

For additional information, see the CoreSight Architecture Specification.

Reads the same value as the [MPIDR_EL1](#) register for the PE that this trace unit has affinity with.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCDEVAFF are RES0.

Attributes

TRCDEVAFF is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | | MPIDR_EL1 | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | | MPIDR_EL1 | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

MPIDR_EL1, bits [63:0]

Read-only copy of [MPIDR_EL1](#), as seen from the highest implemented Exception level.

Accessing TRCDEVAFF

External debugger accesses to this register are unaffected by the OS Lock.

TRCDEVAFF can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| ETE | 0xFA8 | TRCDEVAFF |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVARCH, Device Architecture Register

The TRCDEVARCH characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

External register TRCDEVARCH bits [31:0] are architecturally mapped to AArch64 System register [TRCDEVARCH\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCDEVARCH are RES0.

Attributes

TRCDEVARCH is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------|----|----|----|----|----|----|----|----|----|----|---------|----------|----|----|----|---------|----|----|----|----------|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| ARCHITECT | | | | | | | | | | | PRESENT | REVISION | | | | ARCHVER | | | | ARCHPART | | | | | | | | | | | |

ARCHITECT, bits [31:21]

Architect. Defines the architect of the component. Bits [31:28] are the JEP106 continuation code (JEP106 bank ID, minus 1) and bits [27:21] are the JEP106 ID code.

| ARCHITECT | Meaning |
|---------------|--|
| 0b01000111011 | JEP106 continuation code 0x4, ID code 0x3B. Arm Limited. |

Other values are defined by the JEDEC JEP106 standard.

This field reads as 0x23B.

PRESENT, bit [20]

DEVARCH Present. Defines that the DEVARCH register is present.

| PRESENT | Meaning |
|---------|--|
| 0b0 | Device Architecture information not present. |
| 0b1 | Device Architecture information present. |

This field reads as 1.

REVISION, bits [19:16]

Revision. Defines the architecture revision of the component.

| REVISION | Meaning |
|----------|------------------------|
| 0b0000 | ETEv1.0, FEAT_ETE. |
| 0b0001 | ETEv1.1, FEAT_ETEv1p1. |
| 0b0010 | ETEv1.2, FEAT_ETEv1p2. |

All other values are reserved.

ARCHVER, bits [15:12]

Architecture Version. Defines the architecture version of the component.

| ARCHVER | Meaning |
|---------|---------|
| 0b0101 | ETEv1. |

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHVER is ARCHID[15:12].

This field reads as 0x5.

ARCHPART, bits [11:0]

Architecture Part. Defines the architecture of the component.

| ARCHPART | Meaning |
|----------|----------------------------|
| 0xA13 | Arm PE trace architecture. |

ARCHVER and ARCHPART are also defined as a single field, ARCHID, so that ARCHPART is ARCHID[11:0].

This field reads as 0xA13.

Accessing TRCDEVARCH

External debugger accesses to this register are unaffected by the OS Lock.

TRCDEVARCH can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|------------|
| ETE | 0xFBC | TRCDEVARCH |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCDEVID, Device Configuration Register

The TRCDEVID characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

External register TRCDEVID bits [31:0] are architecturally mapped to AArch64 System register [TRCDEVID\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCDEVID are RES0.

Attributes

TRCDEVID is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |

Bits [31:0]

Reserved, RES0.

Accessing TRCDEVID

External debugger accesses to this register are unaffected by the OS Lock.

TRCDEVID can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFC8 | TRCDEVID |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVID1, Device Configuration Register 1

The TRCDEVID1 characteristics are:

Purpose

Provides discovery information for the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCDEVID1 are RES0.

Attributes

TRCDEVID1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |

Bits [31:0]

Reserved, RES0.

Accessing TRCDEVID1

External debugger accesses to this register are unaffected by the OS Lock.

TRCDEVID1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| ETE | 0xFC4 | TRCDEVID1 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCDEVID2, Device Configuration Register 2

The TRCDEVID2 characteristics are:

Purpose

Provides discovery information for the component.
For additional information, see the CoreSight Architecture Specification.

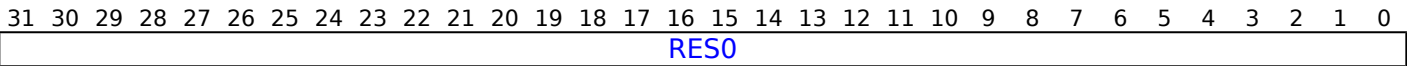
Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCDEVID2 are RES0.

Attributes

TRCDEVID2 is a 32-bit register.

Field descriptions



Bits [31:0]

Reserved, RES0.

Accessing TRCDEVID2

External debugger accesses to this register are unaffected by the OS Lock.

TRCDEVID2 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| ETE | 0xFC0 | TRCDEVID2 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCDEVTYPE, Device Type Register

The TRCDEVTYPE characteristics are:

Purpose

Provides discovery information for the component. If the part number field is not recognized, a debugger can report the information that is provided by TRCDEVTYPE about the component instead.

For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCDEVTYPE are RES0.

Attributes

TRCDEVTYPE is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|-----|---|---|-------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | SUB | | | MAJOR | | | | |

Bits [31:8]

Reserved, RES0.

SUB, bits [7:4]

Component sub-type.

| SUB | Meaning |
|--------|---|
| 0b0001 | When MAJOR == 0x3 (Trace source): Associated with a PE. |

This field reads as 0x1.

MAJOR, bits [3:0]

Component major type.

| MAJOR | Meaning |
|--------|---------------|
| 0b0011 | Trace source. |

Other values are defined by the CoreSight Architecture.

This field reads as 0x3.

Accessing TRCDEVTYPE

External debugger accesses to this register are unaffected by the OS Lock.

TRCDEVTYPE can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
|-----------|--------|----------|

TRCDEVTYPE, Device Type Register

| | | |
|-----|-------|------------|
| ETE | 0xFCC | TRCDEVTYPE |
|-----|-------|------------|

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCEVENTCTL0R, Event Control 0 Register

The TRCEVENTCTL0R characteristics are:

Purpose

Controls the generation of ETEEvents.

Configuration

External register TRCEVENTCTL0R bits [31:0] are architecturally mapped to AArch64 System register [TRCEVENTCTL0R\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMRSPAIR != 0b0000. Otherwise, direct accesses to TRCEVENTCTL0R are RES0.

Attributes

TRCEVENTCTL0R is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-----------------------------|----------------------|----------------------------|-----------------------------|----------------------|----------------------------|-----------------------------|----------------------|----------------------------|-----------------------------|----------------------|----------------------------|-----------------------------|----------------------|----------------------------|-----------------------------|----------------------|----------------------------|-----------------------------|----------------------|----------------------------|-----------------------------|----------------------|----------------------------|-----------------------------|----------------------|----------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 |
| EVENT3_TYPE | RES0 | EVENT3_SEL | EVENT2_TYPE | RES0 | EVENT2_SEL | EVENT1_TYPE | RES0 | EVENT1_SEL | EVENT0_TYPE | RES0 | EVENT0_SEL | EVENT0_TYPE | RES0 | EVENT0_SEL | EVENT0_TYPE | RES0 | EVENT0_SEL | EVENT0_TYPE | RES0 | EVENT0_SEL | EVENT0_TYPE | RES0 | EVENT0_SEL | EVENT0_TYPE | RES0 | EVENT0_SEL |

EVENT3_TYPE, bit [31]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b11:

Chooses the type of Resource Selector.

| EVENT3_TYPE | Meaning |
|-------------|---|
| 0b0 | A single Resource Selector. TRCEVENTCTL0R.EVENT3.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT3.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT3.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [30:29]

Reserved, RES0.

EVENT3_SEL, bits [28:24]**When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b11:**

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT3.TYPE controls whether TRCEVENTCTL0R.EVENT3.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

When any of the selected resource events occurs and [TRCEVENTCTL1R](#).INSTEN[3] == 1, then Event element 3 is generated in the instruction trace element stream.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT2_TYPE, bit [23]**When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b10:**

Chooses the type of Resource Selector.

| EVENT2_TYPE | Meaning |
|-------------|---|
| 0b0 | A single Resource Selector. TRCEVENTCTL0R.EVENT2.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT2.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [22:21]

Reserved, RES0.

EVENT2_SEL, bits [20:16]**When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b10:**

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT2.TYPE controls whether TRCEVENTCTL0R.EVENT2.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

When any of the selected resource events occurs and [TRCEVENTCTL1R](#).INSTEN[2] == 1, then Event element 2 is generated in the instruction trace element stream.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT1_TYPE, bit [15]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b01:

Chooses the type of Resource Selector.

| EVENT1_TYPE | Meaning |
|-------------|---|
| 0b0 | A single Resource Selector. TRCEVENTCTL0R.EVENT1.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT1.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [14:13]

Reserved, RES0.

EVENT1_SEL, bits [12:8]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b01:

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT1.TYPE controls whether TRCEVENTCTL0R.EVENT1.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

When any of the selected resource events occurs and [TRCEVENTCTL1R](#).INSTEN[1] == 1, then Event element 1 is generated in the instruction trace element stream.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EVENT0_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b00:

Chooses the type of Resource Selector.

| EVENT0_TYPE | Meaning |
|-------------|---|
| 0b0 | A single Resource Selector. TRCEVENTCTL0R.EVENT0.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCEVENTCTL0R.EVENT0.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCEVENTCTL0R.EVENT0.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EVENT0_SEL, bits [4:0]

When TRCIDR4.NUMRSPAIR != 0b0000 and TRCIDR0.NUMEVENT >= 0b00:

Defines the selected Resource Selector or pair of Resource Selectors. TRCEVENTCTL0R.EVENT0.TYPE controls whether TRCEVENTCTL0R.EVENT0.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

When any of the selected resource events occurs and [TRCEVENTCTL1R](#).INSTEN[0] == 1, then Event element 0 is generated in the instruction trace element stream.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCEVENTCTL0R

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCEVENTCTL0R can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|---------------|
| ETE | 0x020 | TRCEVENTCTL0R |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCEVENTCTL1R, Event Control 1 Register

The TRCEVENTCTL1R characteristics are:

Purpose

Controls the behavior of the ETEEvents that [TRCEVENTCTL0R](#) selects.

Configuration

External register TRCEVENTCTL1R bits [31:0] are architecturally mapped to AArch64 System register [TRCEVENTCTL1R\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCEVENTCTL1R are RES0.

Attributes

TRCEVENTCTL1R is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|-----|------|-----------|-----------|-----------|-----------|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | LPOVERRIDE | | ATB | RES0 | INSTEN[3] | INSTEN[2] | INSTEN[1] | INSTEN[0] | | | | | | | | |

Bits [31:13]

Reserved, RES0.

LPOVERRIDE, bit [12]

When TRCIDR5.LPOVERRIDE == 1:

Low-power Override Mode select.

| LPOVERRIDE | Meaning |
|------------|--|
| 0b0 | Trace unit Low-power Override Mode is not enabled. That is, the trace unit is permitted to enter low-power state. |
| 0b1 | Trace unit Low-power Override Mode is enabled. That is, entry to a low-power state does not affect the trace unit resources or trace generation. |

Otherwise:

Reserved, RES0.

ATB, bit [11]

When TRCIDR5.ATBTRIG == 1:

AMBA Trace Bus (ATB) trigger enable.

If a CoreSight ATB interface is implemented then when ETEEvent 0 occurs the trace unit sets:

- ATID == 0x7D.
- ATDATA to the value of [TRCTRACEIDR](#).

If the width of ATDATA is greater than the width of [TRCTraceIDR](#).TRACEID then the trace unit zeros the upper ATDATA bits.

If ETEEvent 0 is programmed to occur based on program execution, such as an Address Comparator, the ATB trigger might not be inserted into the ATB stream at the same time as any trace generated by that program execution is output by the trace unit. Typically, the generated trace might be buffered in a trace unit which means that the ATB trigger would be output before the associated trace is output.

If ETEEvent 0 is asserted multiple times in close succession, the trace unit is required to generate an ATB trigger for the first assertion, but might ignore one or more of the subsequent assertions. Arm recommends that the window in which ETEEvent 0 is ignored is limited only by the time taken to output an ATB trigger.

| ATB | Meaning |
|-----|--------------------------|
| 0b0 | ATB trigger is disabled. |
| 0b1 | ATB trigger is enabled. |

Otherwise:

Reserved, RES0.

Bits [10:4]

Reserved, RES0.

INSTEN[<m>], bit [m], for m = 3 to 0

Event element control.

| INSTEN[<m>] | Meaning |
|-------------|--|
| 0b0 | The trace unit does not generate an Event element <m>. |
| 0b1 | The trace unit generates an Event element <m>. |

This bit is RES0 if m >= the number indicated by [TRCIDR0](#).NUMEVENT.

Accessing TRCEVENTCTL1R

Must be programmed.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCEVENTCTL1R can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|---------------|
| ETE | 0x024 | TRCEVENTCTL1R |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

TRCEXTINSELR<n>, External Input Select Register <n>, n = 0 - 3

The TRCEXTINSELR<n> characteristics are:

Purpose

Use this to set, or read, which External Inputs are resources to the trace unit.

The name TRCEXTINSELR is an alias of TRCEXTINSELR0.

Configuration

External register TRCEXTINSELR<n> bits [31:0] are architecturally mapped to AArch64 System register [TRCEXTINSELR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMEXTINSEL > n. Otherwise, direct accesses to TRCEXTINSELR<n> are RES0.

Attributes

TRCEXTINSELR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | evtCount | | | | | | | | | | | | | | | |

Bits [31:16]

Reserved, RES0.

evtCount, bits [15:0]

PMU event to select.

The event number as defined by the Arm ARM.

Software must program this field with a PMU event that is supported by the PE being programmed.

There are three ranges of PMU event numbers:

- PMU event numbers in the range 0x0000 to 0x003F are common architectural and microarchitectural events.
- PMU event numbers in the range 0x0040 to 0x00BF are Arm recommended common architectural and microarchitectural PMU events.
- PMU event numbers in the range 0x00C0 to 0x03FF are IMPLEMENTATION DEFINED PMU events.

If evtCount is programmed to a PMU event that is reserved or not supported by the PE, the behavior depends on the PMU event type:

- For the range 0x0000 to 0x003F, then the PMU event is not active, and the value returned by a direct or external read of the evtCount field is the value written to the field.
- For IMPLEMENTATION DEFINED PMU events, it is UNPREDICTABLE what PMU event, if any, is counted, and the value returned by a direct or external read of the evtCount field is UNKNOWN.

UNPREDICTABLE means the PMU event must not expose privileged information.

Arm recommends that the behavior across a family of implementations is defined such that if a given implementation does not include a PMU event from a set of common IMPLEMENTATION DEFINED PMU events, then no PMU event is counted and the value read back on evtCount is the value written.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCEXTINSELN<n>

Must be programmed if any of the following is true: [TRCRSCTLN<a>](#).GROUP == 0b0000 and [TRCRSCTLN<a>](#).EXTIN[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCEXTINSELN<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|-----------------|
| ETE | 0x120 + (4 * n) | TRCEXTINSELN<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR0, ID Register 0

The TRCIDR0 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR0 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR0\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR0 are RES0.

Attributes

TRCIDR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | |
|------|-----------|---------|--------|--------|------|-----------|-------|-------|----------|----------|----------|----|----|----|----|----|----|----|----|----|----|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 |
| RES0 | COMMTRANS | COMMOPT | TSSIZE | TSMARK | RES0 | TRCEXDATA | QSUPP | QFILT | CONDTYPE | NUMEVENT | RETSTACK | | | | | | | | | | | |

Bit [31]

Reserved, RES0.

COMMTRANS, bit [30]

Transaction Start element behavior.

| COMMTRANS | Meaning |
|-----------|---|
| 0b0 | Transaction Start elements are P0 elements. |
| 0b1 | Transaction Start elements are not P0 elements. |

COMMOPT, bit [29]

Indicates the contents and encodings of Cycle count packets.

| COMMOPT | Meaning |
|---------|----------------|
| 0b0 | Commit mode 0. |
| 0b1 | Commit mode 1. |

The Commit mode defines the contents and encodings of Cycle Count packets, in particular how Commit elements are indicated by these packets. See the descriptions of these packets for more details.

Accessing this field has the following behavior:

- **RAO/WI** if all of the following are true:
 - TRCIDR0.TRCCCI == 1
 - TRCIDR8.MAXSPEC == 0x0
- When TRCIDR0.TRCCCI == 0, access to this field is **RAZ/WI**.
- Otherwise, access to this field is **RO**.

TSSIZE, bits [28:24]

Indicates that the trace unit implements Global timestamping and the size of the timestamp value.

| TSSIZE | Meaning |
|---------------|--|
| 0b000000 | Global timestamping not implemented. |
| 0b01000 | Global timestamping implemented with a 64-bit timestamp value. |

All other values are reserved.

This field reads as 0b01000.

TSMARK, bit [23]

When FEAT_ETEv1p1 is implemented:

Indicates whether Timestamp Marker elements are generated.

| TSMARK | Meaning |
|---------------|--|
| 0b0 | Timestamp Marker elements are not generated. |
| 0b1 | Timestamp Marker elements are generated. |

Otherwise:

Reserved, RES0.

Bits [22:18]

Reserved, RES0.

TRCEXDATA, bit [17]

When TRCIDR0.TRCDATA != 0b00:

Indicates if the trace unit implements tracing of data transfers for exceptions and exception returns. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

| TRCEXDATA | Meaning |
|------------------|---|
| 0b0 | Tracing of data transfers for exceptions and exception returns not implemented. |
| 0b1 | Tracing of data transfers for exceptions and exception returns implemented. |

Otherwise:

Reserved, RES0.

QSUPP, bits [16:15]

Indicates that the trace unit implements Q element support.

| QSUPP | Meaning |
|--------------|---|
| 0b00 | Q element support is not implemented. |
| 0b01 | Q element support is implemented, and only supports Q elements with instruction counts. |
| 0b10 | Q element support is implemented, and only supports Q elements without instruction counts. |
| 0b11 | Q element support is implemented, and supports: <ul style="list-style-type: none"> Q elements with instruction counts. Q elements without instruction counts. |

QFILT, bit [14]

Indicates if the trace unit implements Q element filtering.

| QFILT | Meaning |
|-------|---|
| 0b0 | Q element filtering is not implemented. |
| 0b1 | Q element filtering is implemented. |

If TRCIDR0.QSUPP == 0b00 then this field is 0.

CONDTYPE, bits [13:12]

When TRCIDR0.TRCOND == 1:

Indicates how conditional instructions are traced. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

| CONDTYPE | Meaning |
|----------|---|
| 0b00 | Conditional instructions are traced with an indication of whether they pass or fail their condition code check. |
| 0b01 | Conditional instructions are traced with an indication of the APSR condition flags. |

All other values are reserved.

Otherwise:

Reserved, RES0.

NUMEVENT, bits [11:10]

When TRCIDR4.NUMRSPAIR == 0b0000:

Indicates the number of ETEEvents implemented.

| NUMEVENT | Meaning |
|----------|--------------------------------------|
| 0b00 | The trace unit supports 0 ETEEvents. |

All other values are reserved.

When TRCIDR4.NUMRSPAIR != 0b0000:

Indicates the number of ETEEvents implemented.

| NUMEVENT | Meaning |
|----------|--------------------------------------|
| 0b00 | The trace unit supports 1 ETEEvent. |
| 0b01 | The trace unit supports 2 ETEEvents. |
| 0b10 | The trace unit supports 3 ETEEvents. |
| 0b11 | The trace unit supports 4 ETEEvents. |

Otherwise:

Reserved, RES0.

RETSTACK, bit [9]

Indicates if the trace unit supports the return stack.

| RETSTACK | Meaning |
|----------|-------------------------------|
| 0b0 | Return stack not implemented. |
| 0b1 | Return stack implemented. |

Bit [8]

Reserved, RES0.

TRCCCI, bit [7]

Indicates if the trace unit implements cycle counting.

| TRCCCI | Meaning |
|--------|---------------------------------|
| 0b0 | Cycle counting not implemented. |
| 0b1 | Cycle counting implemented. |

This field reads as 1.

TRCCOND, bit [6]

Indicates if the trace unit implements conditional instruction tracing. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures.

| TRCCOND | Meaning |
|---------|--|
| 0b0 | Conditional instruction tracing not implemented. |
| 0b1 | Conditional instruction tracing implemented. |

This field reads as 0.

TRCBB, bit [5]

Indicates if the trace unit implements branch broadcasting.

| TRCBB | Meaning |
|-------|--------------------------------------|
| 0b0 | Branch broadcasting not implemented. |
| 0b1 | Branch broadcasting implemented. |

This field reads as 1.

TRCDATA, bits [4:3]

Indicates if the trace unit implements data tracing. Data tracing is not implemented in ETE and this field is reserved for other trace architectures.

| TRCDATA | Meaning |
|---------|-------------------------------|
| 0b00 | Data tracing not implemented. |
| 0b11 | Data tracing implemented. |

All other values are reserved.

This field reads as 0b00.

INSTP0, bits [2:1]

Indicates if load and store instructions are P0 instructions. Load and store instructions as P0 instructions is not implemented in ETE and this field is reserved for other trace architectures.

| INSTP0 | Meaning |
|--------|--|
| 0b00 | Load and store instructions are not P0 instructions. |
| 0b11 | Load and store instructions are P0 instructions. |

All other values are reserved.

This field reads as 0b00.

Bit [0]

Reserved, RES1.

Accessing TRCIDR0

TRCIDR0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x1E0 | TRCIDR0 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR1, ID Register 1

The TRCIDR1 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR1 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR1\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR1 are RES0.

Attributes

TRCIDR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------------|----|----|----|----|----|----|----|----------------------|----|----|----|----|----|----|----|----------------------|----|----|----|----------------------------|----|---|---|----------------------------|---|---|---|--------------------------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| DESIGNER | | | | | | | | RES0 | | | | | | | | RES1 | | | | TRCARCHMAJ | | | | TRCARCHMIN | | | | REVISION | | | |

DESIGNER, bits [31:24]

Indicates which company designed the trace unit. The permitted values of this field are the same as [MIDR_EL1](#).Implementer.

Bits [23:16]

Reserved, RES0.

Bits [15:12]

Reserved, RES1.

TRCARCHMAJ, bits [11:8]

Major architecture version.

| TRCARCHMAJ | Meaning |
|------------|---|
| 0b1111 | If both TRCARCHMAJ and TRCARCHMIN == 0xF then refer to TRCDEVARCH . |

All other values are reserved.

This field reads as 0b1111.

TRCARCHMIN, bits [7:4]

Minor architecture version.

| TRCARCHMIN | Meaning |
|------------|---|
| 0b1111 | If both TRCARCHMAJ and TRCARCHMIN == 0xF then refer to TRCDEVARCH . |

All other values are reserved.

This field reads as 0b1111.

REVISION, bits [3:0]

Implementation revision.

Returns an IMPLEMENTATION DEFINED value that identifies the revision of the trace unit.

Arm deprecates any use of this field and recommends that implementations set this field to zero.

Accessing TRCIDR1

TRCIDR1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x1E4 | TRCIDR1 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR10, ID Register 10

The TRCIDR10 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR10 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR10\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR10 are RES0.

Attributes

TRCIDR10 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NUMP1KEY | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

NUMP1KEY, bits [31:0]

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

Accessing TRCIDR10

TRCIDR10 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x188 | TRCIDR10 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCIDR11, ID Register 11

The TRCIDR11 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR11 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR11\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR11 are RES0.

Attributes

TRCIDR11 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NUMP1SPC | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

NUMP1SPC, bits [31:0]
When TRCIDR0.TRCDATA != 0b00:

Indicates the number of special P1 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

Accessing TRCIDR11

TRCIDR11 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x18C | TRCIDR11 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCIDR12, ID Register 12

The TRCIDR12 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR12 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR12\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR12 are RES0.

Attributes

TRCIDR12 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NUMCONDKEY | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

NUMCONDKEY, bits [31:0]

When TRCIDR0.TRCCOND == 1:

Indicates the number of conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

Accessing TRCIDR12

TRCIDR12 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x190 | TRCIDR12 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCIDR13, ID Register 13

The TRCIDR13 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR13 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR13\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR13 are RES0.

Attributes

TRCIDR13 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NUMCONDSPC | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

NUMCONDSPC, bits [31:0]

When TRCIDR0.TRCCOND == 1:

Indicates the number of special conditional instruction right-hand keys. Conditional instruction tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

Accessing TRCIDR13

TRCIDR13 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x194 | TRCIDR13 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCIDR2, ID Register 2

The TRCIDR2 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR2 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR2\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR2 are RES0.

Attributes

TRCIDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|-------------------------|-------------------------|----|------------------------|----|----|----|----|------------------------|----|----|----|----|------------------------|----|----|----|--------------------------|----|----|----|----|-------------------------|---|---|---|---|---|---|---|---|------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| WFXMODE | VMIDOPT | | CCSIZE | | | | | DVSIZE | | | | | DASIZE | | | | VMIDSIZE | | | | | CIDSIZE | | | | | | | | | IASIZE |

WFXMODE, bit [31]

Indicates whether WFI and WFE instructions are classified as P0 instructions:

| WFXMODE | Meaning |
|---------|---|
| 0b0 | WFI and WFE instructions are not classified as P0 instructions. |
| 0b1 | WFI and WFE instructions are classified as P0 instructions. |

VMIDOPT, bits [30:29]

Indicates the options for Virtual context identifier selection.

| VMIDOPT | Meaning |
|---------|--|
| 0b00 | Virtual context identifier selection not supported. TRCCONFIGR .VMIDOPT is RES0. |
| 0b01 | Virtual context identifier selection supported. TRCCONFIGR .VMIDOPT is implemented. |
| 0b10 | Virtual context identifier selection not supported. TRCCONFIGR .VMIDOPT is RES1. |

All other values are reserved.

If TRCIDR2.VMIDSIZE == 0b000000 then this field is 0b00.

If TRCIDR2.VMIDSIZE != 0b000000 then this field is 0b10.

CCSIZE, bits [28:25]

When TRCIDR0.TRCCCI == 1:

Indicates the size of the cycle counter.

| CCSIZE | Meaning |
|---------------|---|
| 0b0000 | The cycle counter is 12 bits in length. |
| 0b0001 | The cycle counter is 13 bits in length. |
| 0b0010 | The cycle counter is 14 bits in length. |
| 0b0011 | The cycle counter is 15 bits in length. |
| 0b0100 | The cycle counter is 16 bits in length. |
| 0b0101 | The cycle counter is 17 bits in length. |
| 0b0110 | The cycle counter is 18 bits in length. |
| 0b0111 | The cycle counter is 19 bits in length. |
| 0b1000 | The cycle counter is 20 bits in length. |

All other values are reserved.

Otherwise:

Reserved, RES0.

DVSIZE, bits [24:20]

When TRCIDR0.TRCDATA != 0b00:

Indicates the data value size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

| DVSIZE | Meaning |
|---------------|---|
| 0b00000 | Data value tracing not implemented. |
| 0b00100 | Data value tracing has a maximum of 32-bit data values. |
| 0b01000 | Data value tracing has a maximum of 64-bit data values. |

All other values are reserved.

Otherwise:

Reserved, RES0.

DASIZE, bits [19:15]

When TRCIDR0.TRCDATA != 0b00:

Indicates the data address size in bytes. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

| DASIZE | Meaning |
|---------------|--|
| 0b00000 | Data address tracing not implemented. |
| 0b00100 | Data address tracing has a maximum of 32-bit data addresses. |
| 0b01000 | Data address tracing has a maximum of 64-bit data addresses. |

All other values are reserved.

Otherwise:

Reserved, RES0.

VMIDSIZE, bits [14:10]

Indicates the trace unit Virtual context identifier size.

| VMIDSIZE | Meaning |
|-----------------|--|
| 0b00000 | Virtual context identifier tracing is not supported. |
| 0b00001 | 8-bit Virtual context identifier size. |
| 0b00010 | 16-bit Virtual context identifier size. |
| 0b00100 | 32-bit Virtual context identifier size. |

All other values are reserved.

If the PE does not implement EL2 then this field is 0b00000.

If the PE implements EL2 then this field is 0b00100.

CIDSIZE, bits [9:5]

Indicates the Context identifier size.

| CIDSIZE | Meaning |
|----------------|--|
| 0b00000 | Context identifier tracing is not supported. |
| 0b00100 | 32-bit Context identifier size. |

All other values are reserved.

This field reads as 0b00100.

IASIZE, bits [4:0]

Virtual instruction address size.

| IASIZE | Meaning |
|---------------|---|
| 0b00100 | Maximum of 32-bit instruction address size. |
| 0b01000 | Maximum of 64-bit instruction address size. |

All other values are reserved.

This field reads as 0b01000.

Accessing TRCIDR2

TRCIDR2 can be accessed through the external debug interface:

| Component | Offset | Instance |
|------------------|---------------|-----------------|
| ETE | 0x1E8 | TRCIDR2 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCIDR3, ID Register 3

The TRCIDR3 characteristics are:

Purpose

Returns the base architecture of the trace unit.

Configuration

External register TRCIDR3 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR3\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR3 are RES0.

Attributes

TRCIDR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | |
|----------------------------|------------------------------|--------------------------|--------------------------|------------------------|------------------------|----------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|--------------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| NOOVERFLOW | NUMPROC[2:0] | SYSSTALL | STALLCTL | SYNCPR | TRCERR | RES0 | EXLEVEL_NS_EL2 | EXLEVEL_NS_EL1 | EXLEVEL_NS_EL0 | EXLEVEL_NS_EL0 | EXLEVEL_NS_EL0 |

NOOVERFLOW, bit [31]

Indicates if overflow prevention is implemented.

| NOOVERFLOW | Meaning |
|------------|---|
| 0b0 | Overflow prevention is not implemented. |
| 0b1 | Overflow prevention is implemented. |

If TRCIDR3.STALLCTL == 0 then this field is 0.

NUMPROC, bits [13:12, 30:28]

Indicates the number of PEs available for tracing.

| NUMPROC | Meaning |
|---------|----------------------------------|
| 0b00000 | The trace unit can trace one PE. |

This field reads as 0b00000.

The NUMPROC field is split as follows:

- NUMPROC[2:0] is TRCIDR3[30:28].
- NUMPROC[4:3] is TRCIDR3[13:12].

SYSSTALL, bit [27]

Indicates if stalling of the PE is permitted.

| SYSSTALL | Meaning |
|----------|--------------------------------------|
| 0b0 | Stalling of the PE is not permitted. |
| 0b1 | Stalling of the PE is permitted. |

The value of this field might be dynamic and change based on system conditions.

If TRCIDR3.STALLCTL == 0 then this field is 0.

STALLCTL, bit [26]

Indicates if trace unit implements stalling of the PE.

| STALLCTL | Meaning |
|----------|--|
| 0b0 | Stalling of the PE is not implemented. |
| 0b1 | Stalling of the PE is implemented. |

SYNCPR, bit [25]

Indicates if an implementation has a fixed synchronization period.

| SYNCPR | Meaning |
|--------|--|
| 0b0 | TRCSYNCPR is read-write so software can change the synchronization period. |
| 0b1 | TRCSYNCPR is read-only so the synchronization period is fixed. |

This field reads as 0.

TRCERR, bit [24]

Indicates forced tracing of System Error exceptions is implemented.

| TRCERR | Meaning |
|--------|---|
| 0b0 | Forced tracing of System Error exceptions is not implemented. |
| 0b1 | Forced tracing of System Error exceptions is implemented. |

This field reads as 1.

Bit [23]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [22]

Indicates if Non-secure EL2 is implemented.

| EXLEVEL_NS_EL2 | Meaning |
|----------------|------------------------------------|
| 0b0 | Non-secure EL2 is not implemented. |
| 0b1 | Non-secure EL2 is implemented. |

EXLEVEL_NS_EL1, bit [21]

Indicates if Non-secure EL1 is implemented.

| EXLEVEL_NS_EL1 | Meaning |
|----------------|------------------------------------|
| 0b0 | Non-secure EL1 is not implemented. |
| 0b1 | Non-secure EL1 is implemented. |

EXLEVEL_NS_EL0, bit [20]

Indicates if Non-secure EL0 is implemented.

| EXLEVEL_NS_EL0 | Meaning |
|----------------|------------------------------------|
| 0b0 | Non-secure EL0 is not implemented. |
| 0b1 | Non-secure EL0 is implemented. |

EXLEVEL_S_EL3, bit [19]

Indicates if EL3 is implemented.

| EXLEVEL_S_EL3 | Meaning |
|---------------|-------------------------|
| 0b0 | EL3 is not implemented. |
| 0b1 | EL3 is implemented. |

EXLEVEL_S_EL2, bit [18]

Indicates if Secure EL2 is implemented.

| EXLEVEL_S_EL2 | Meaning |
|---------------|--------------------------------|
| 0b0 | Secure EL2 is not implemented. |
| 0b1 | Secure EL2 is implemented. |

EXLEVEL_S_EL1, bit [17]

Indicates if Secure EL1 is implemented.

| EXLEVEL_S_EL1 | Meaning |
|---------------|--------------------------------|
| 0b0 | Secure EL1 is not implemented. |
| 0b1 | Secure EL1 is implemented. |

EXLEVEL_S_EL0, bit [16]

Indicates if Secure EL0 is implemented.

| EXLEVEL_S_EL0 | Meaning |
|---------------|--------------------------------|
| 0b0 | Secure EL0 is not implemented. |
| 0b1 | Secure EL0 is implemented. |

Bits [15:14]

Reserved, RES0.

CCITMIN, bits [11:0]

Indicates the minimum value that can be programmed in [TRCCCCTLR.THRESHOLD](#).

If [TRCIDR0.TRCCCI](#) == 1 then the minimum value of this field is 0x001.

If [TRCIDR0.TRCCCI](#) == 0 then this field is zero.

Accessing TRCIDR3

TRCIDR3 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x1EC | TRCIDR3 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCIDR4, ID Register 4

The TRCIDR4 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR4 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR4\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR4 are RES0.

Attributes

TRCIDR4 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------------------------|-------------------------|-------------------------|---------------------------|-----------------------|----------------------|-------------------------|------------------------|----------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NUMVMIDC | NUMCIDC | NUMSSCC | NUMRSPAIR | NUMPC | RES0 | SUPPDAC | NUMDVC | NUMACPAIRS | | | | | | | | | | | | | | | | | | | | | | | |

NUMVMIDC, bits [31:28]

Indicates the number of Virtual Context Identifier Comparators that are available for tracing.

| NUMVMIDC | Meaning |
|----------|--|
| 0b0000 | No Virtual Context Identifier Comparators are available. |
| 0b0001 | The implementation has one Virtual Context Identifier Comparator. |
| 0b0010 | The implementation has two Virtual Context Identifier Comparators. |
| 0b0011 | The implementation has three Virtual Context Identifier Comparators. |
| 0b0100 | The implementation has four Virtual Context Identifier Comparators. |
| 0b0101 | The implementation has five Virtual Context Identifier Comparators. |
| 0b0110 | The implementation has six Virtual Context Identifier Comparators. |
| 0b0111 | The implementation has seven Virtual Context Identifier Comparators. |
| 0b1000 | The implementation has eight Virtual Context Identifier Comparators. |

All other values are reserved.

NUMCIDC, bits [27:24]

Indicates the number of Context Identifier Comparators that are available for tracing.

| NUMC IDC | Meaning |
|----------|--|
| 0b0000 | No Context Identifier Comparators are available. |
| 0b0001 | The implementation has one Context Identifier Comparator. |
| 0b0010 | The implementation has two Context Identifier Comparators. |
| 0b0011 | The implementation has three Context Identifier Comparators. |
| 0b0100 | The implementation has four Context Identifier Comparators. |
| 0b0101 | The implementation has five Context Identifier Comparators. |
| 0b0110 | The implementation has six Context Identifier Comparators. |
| 0b0111 | The implementation has seven Context Identifier Comparators. |
| 0b1000 | The implementation has eight Context Identifier Comparators. |

All other values are reserved.

NUMSSCC, bits [23:20]

Indicates the number of Single-shot Comparator Controls that are available for tracing.

| NUMSSCC | Meaning |
|---------|---|
| 0b0000 | No Single-shot Comparator Controls are available. |
| 0b0001 | The implementation has one Single-shot Comparator Control. |
| 0b0010 | The implementation has two Single-shot Comparator Controls. |
| 0b0011 | The implementation has three Single-shot Comparator Controls. |
| 0b0100 | The implementation has four Single-shot Comparator Controls. |
| 0b0101 | The implementation has five Single-shot Comparator Controls. |
| 0b0110 | The implementation has six Single-shot Comparator Controls. |
| 0b0111 | The implementation has seven Single-shot Comparator Controls. |
| 0b1000 | The implementation has eight Single-shot Comparator Controls. |

All other values are reserved.

NUMRSPAIR, bits [19:16]

Indicates the number of resource selector pairs that are available for tracing.

| NUMRSPAIR | Meaning |
|-----------|--|
| 0b0000 | The implementation has zero resource selectors. |
| 0b0001 | The implementation has two resource selector pairs. |
| 0b0010 | The implementation has three resource selector pairs. |
| 0b0011 | The implementation has four resource selector pairs. |
| 0b0100 | The implementation has five resource selector pairs. |
| 0b0101 | The implementation has six resource selector pairs. |
| 0b0110 | The implementation has seven resource selector pairs. |
| 0b0111 | The implementation has eight resource selector pairs. |
| 0b1000 | The implementation has nine resource selector pairs. |
| 0b1001 | The implementation has ten resource selector pairs. |
| 0b1010 | The implementation has eleven resource selector pairs. |
| 0b1011 | The implementation has twelve resource selector pairs. |
| 0b1100 | The implementation has thirteen resource selector pairs. |
| 0b1101 | The implementation has fourteen resource selector pairs. |
| 0b1110 | The implementation has fifteen resource selector pairs. |
| 0b1111 | The implementation has sixteen resource selector pairs. |

All other values are reserved.

NUMPC, bits [15:12]

Indicates the number of PE Comparator Inputs that are available for tracing.

| NUMPC | Meaning |
|--------|--|
| 0b0000 | No PE Comparator Inputs are available. |
| 0b0001 | The implementation has one PE Comparator Input. |
| 0b0010 | The implementation has two PE Comparator Inputs. |
| 0b0011 | The implementation has three PE Comparator Inputs. |
| 0b0100 | The implementation has four PE Comparator Inputs. |
| 0b0101 | The implementation has five PE Comparator Inputs. |
| 0b0110 | The implementation has six PE Comparator Inputs. |
| 0b0111 | The implementation has seven PE Comparator Inputs. |
| 0b1000 | The implementation has eight PE Comparator Inputs. |

All other values are reserved.

Bits [11:9]

Reserved, RES0.

SUPPDAC, bit [8]

When TRCIDR4.NUMACPAIRS != 0b0000:

Indicates whether data address comparisons are implemented. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

| SUPPDAC | Meaning |
|---------|---|
| 0b0 | Data address comparisons not implemented. |
| 0b1 | Data address comparisons implemented. |

This field reads as 0.

Otherwise:

Reserved, RES0.

NUMDVC, bits [7:4]

Indicates the number of data value comparators. Data value comparators are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

| NUMDVC | Meaning |
|--------|---|
| 0b0000 | No data value comparators implemented. |
| 0b0001 | One data value comparator implemented. |
| 0b0010 | Two data value comparators implemented. |
| 0b0011 | Three data value comparators implemented. |
| 0b0100 | Four data value comparators implemented. |
| 0b0101 | Five data value comparators implemented. |
| 0b0110 | Six data value comparators implemented. |
| 0b0111 | Seven data value comparators implemented. |
| 0b1000 | Eight data value comparators implemented. |

All other values are reserved.

This field reads as 0b0000.

NUMACPAIRS, bits [3:0]

Indicates the number of Address Comparator pairs that are available for tracing.

| NUMACPAIRS | Meaning |
|------------|--|
| 0b0000 | No Address Comparator pairs are available. |
| 0b0001 | The implementation has one Address Comparator pair. |
| 0b0010 | The implementation has two Address Comparator pairs. |
| 0b0011 | The implementation has three Address Comparator pairs. |
| 0b0100 | The implementation has four Address Comparator pairs. |
| 0b0101 | The implementation has five Address Comparator pairs. |
| 0b0110 | The implementation has six Address Comparator pairs. |
| 0b0111 | The implementation has seven Address Comparator pairs. |
| 0b1000 | The implementation has eight Address Comparator pairs. |

All other values are reserved.

Accessing TRCIDR4

TRCIDR4 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x1F0 | TRCIDR4 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCIDR5, ID Register 5

The TRCIDR5 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR5 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR5\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR5 are RES0.

Attributes

TRCIDR5 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|----------------------|----------------------------|-------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|----------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|-----------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | NUMCNTR | NUMCNTR | NUMCNTR | NUMCNTR | NUMCNTR | NUMCNTR | NUMCNTR | RES0 | LPOVERRIDE | ATBTRIG | TRACEIDSIZE | TRACEIDSIZE | TRACEIDSIZE | TRACEIDSIZE | TRACEIDSIZE | TRACEIDSIZE | TRACEIDSIZE | TRACEIDSIZE | TRACEIDSIZE | RES0 | NUMEXTINSEL | NUMEXTINSEL | NUMEXTINSEL | NUMEXTINSEL | NUMEXTINSEL | NUMEXTINSEL | NUMEXTINSEL | NUMEXTINSEL | NUMEXTINSEL | NUMEXTINSEL | NUMEXTINSEL |

Bit [31]

Reserved, RES0.

NUMCNTR, bits [30:28]

Indicates the number of Counters that are available for tracing.

| NUMCNTR | Meaning |
|---------|-----------------------------|
| 0b000 | No Counters are available. |
| 0b001 | One Counter implemented. |
| 0b010 | Two Counters implemented. |
| 0b011 | Three Counters implemented. |
| 0b100 | Four Counters implemented. |

All other values are reserved.

If [TRCIDR4](#).NUMRSPAIR == 0b0000 then this field is 0b000.

NUMSEQSTATE, bits [27:25]

Indicates if the Sequencer is implemented and the number of Sequencer states that are implemented.

| NUMSEQSTATE | Meaning |
|-------------|--|
| 0b000 | The Sequencer is not implemented. |
| 0b100 | Four Sequencer states are implemented. |

All other values are reserved.

If [TRCIDR4](#).NUMRSPAIR == 0b0000 then this field is 0b000.

Bit [24]

Reserved, RES0.

LPOVERRIDE, bit [23]

Indicates support for Low-power Override Mode.

| LPOVERRIDE | Meaning |
|-------------------|--|
| 0b0 | The trace unit does not support Low-power Override Mode. |
| 0b1 | The trace unit supports Low-power Override Mode. |

ATBTRIG, bit [22]

Indicates if the implementation can support ATB triggers.

| ATBTRIG | Meaning |
|----------------|---|
| 0b0 | The implementation does not support ATB triggers. |
| 0b1 | The implementation supports ATB triggers. |

If [TRCIDR4](#).NUMRSPAIR == 0b0000 then this field is 0.

TRACEIDSIZE, bits [21:16]

Indicates the trace ID width.

| TRACEIDSIZE | Meaning |
|--------------------|--|
| 0b000000 | The external trace interface is not implemented. |
| 0b000111 | The implementation supports a 7-bit trace ID. |

All other values are reserved.

Note that AMBA ATB requires a 7-bit trace ID width.

Bits [15:12]

Reserved, RES0.

NUMEXTINSEL, bits [11:9]

Indicates how many External Input Selector resources are implemented.

| NUMEXTINSEL | Meaning |
|--------------------|---|
| 0b000 | No External Input Selector resources are available. |
| 0b001 | 1 External Input Selector resource is available. |
| 0b010 | 2 External Input Selector resources are available. |
| 0b011 | 3 External Input Selector resources are available. |
| 0b100 | 4 External Input Selector resources are available. |

All other values are reserved.

NUMEXTIN, bits [8:0]

Indicates how many External Inputs are implemented.

| NUMEXTIN | Meaning |
|-----------------|------------------------------|
| 0b11111111 | Unified PMU event selection. |

All other values are reserved.

Accessing TRCIDR5

TRCIDR5 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x1F4 | TRCIDR5 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR6, ID Register 6

The TRCIDR6 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR6 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR6\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR6 are RES0.

Attributes

TRCIDR6 is a 32-bit register.

Field descriptions

| | | | |
|---|--------------------------------|--------------------------------|--------------------------------|
| 313029282726252423222120191817161514131211109876543 | 2 | 1 | 0 |
| RES0 | EXLEVEL_RL_EL2 | EXLEVEL_RL_EL1 | EXLEVEL_RL_EL0 |

Bits [31:3]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [2]

Indicates if Realm EL2 is implemented.

| EXLEVEL_RL_EL2 | Meaning |
|----------------|-------------------------------|
| 0b0 | Realm EL2 is not implemented. |
| 0b1 | Realm EL2 is implemented. |

EXLEVEL_RL_EL1, bit [1]

Indicates if Realm EL1 is implemented.

| EXLEVEL_RL_EL1 | Meaning |
|----------------|-------------------------------|
| 0b0 | Realm EL1 is not implemented. |
| 0b1 | Realm EL1 is implemented. |

EXLEVEL_RL_EL0, bit [0]

Indicates if Realm EL0 is implemented.

| EXLEVEL_RL_EL0 | Meaning |
|----------------|-------------------------------|
| 0b0 | Realm EL0 is not implemented. |
| 0b1 | Realm EL0 is implemented. |

Accessing TRCIDR6

TRCIDR6 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x1F8 | TRCIDR6 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR7, ID Register 7

The TRCIDR7 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR7 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR7\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR7 are RES0.

Attributes

TRCIDR7 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |

Bits [31:0]

Reserved, RES0.

Accessing TRCIDR7

TRCIDR7 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x1FC | TRCIDR7 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR8, ID Register 8

The TRCIDR8 characteristics are:

Purpose

Returns the maximum speculation depth of the instruction trace element stream.

Configuration

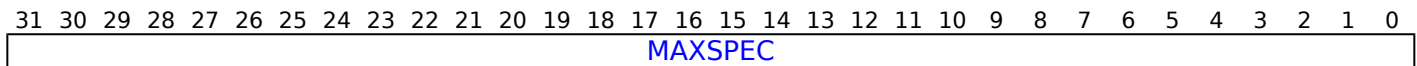
External register TRCIDR8 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR8\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR8 are RES0.

Attributes

TRCIDR8 is a 32-bit register.

Field descriptions



MAXSPEC, bits [31:0]

Indicates the maximum speculation depth of the instruction trace element stream. This is the maximum number of P0 elements in the trace element stream that can be speculative at any time.

Accessing TRCIDR8

TRCIDR8 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x180 | TRCIDR8 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIDR9, ID Register 9

The TRCIDR9 characteristics are:

Purpose

Returns the tracing capabilities of the trace unit.

Configuration

External register TRCIDR9 bits [31:0] are architecturally mapped to AArch64 System register [TRCIDR9\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIDR9 are RES0.

Attributes

TRCIDR9 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| NUMPOKEY | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

NUMPOKEY, bits [31:0]

When TRCIDR0.TRCDATA != 0b00:

Indicates the number of P0 right-hand keys. Data tracing is not implemented in ETE and this field is reserved for other trace architectures. Allocated in other trace architectures.

Otherwise:

Reserved, RES0.

Accessing TRCIDR9

TRCIDR9 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x184 | TRCIDR9 |

This interface is accessible as follows:

- When OSLockStatus() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCIMSPEC0, IMP DEF Register 0

The TRCIMSPEC0 characteristics are:

Purpose

TRCIMSPEC0 shows the presence of any IMPLEMENTATION DEFINED features, and provides an interface to enable the features that are provided.

Configuration

External register TRCIMSPEC0 bits [31:0] are architecturally mapped to AArch64 System register [TRCIMSPEC0\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCIMSPEC0 are RES0.

Attributes

TRCIMSPEC0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|----|---|---------|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | EN | | SUPPORT | | | | | |

Bits [31:8]

Reserved, RES0.

EN, bits [7:4]

When TRCIMSPEC0.SUPPORT != 0b0000:

Enable. Controls whether the IMPLEMENTATION DEFINED features are enabled.

| EN | Meaning |
|--------|--|
| 0b0000 | The IMPLEMENTATION DEFINED features are not enabled. The trace unit must behave as if the IMPLEMENTATION DEFINED features are not supported. |
| 0b0001 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b0010 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b0011 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b0100 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b0101 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b0110 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b0111 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1000 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1001 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1010 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1011 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1100 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1101 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1110 | The trace unit behavior is IMPLEMENTATION DEFINED. |
| 0b1111 | The trace unit behavior is IMPLEMENTATION DEFINED. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to 0.

Otherwise:

Reserved, RES0.

SUPPORT, bits [3:0]

Indicates whether the implementation supports IMPLEMENTATION DEFINED features.

| SUPPORT | Meaning |
|---------|---|
| 0b0000 | No IMPLEMENTATION DEFINED features are supported. |
| 0b0001 | IMPLEMENTATION DEFINED features are supported. |
| 0b0010 | IMPLEMENTATION DEFINED features are supported. |
| 0b0011 | IMPLEMENTATION DEFINED features are supported. |
| 0b0100 | IMPLEMENTATION DEFINED features are supported. |
| 0b0101 | IMPLEMENTATION DEFINED features are supported. |
| 0b0110 | IMPLEMENTATION DEFINED features are supported. |
| 0b0111 | IMPLEMENTATION DEFINED features are supported. |
| 0b1000 | IMPLEMENTATION DEFINED features are supported. |
| 0b1001 | IMPLEMENTATION DEFINED features are supported. |
| 0b1010 | IMPLEMENTATION DEFINED features are supported. |
| 0b1011 | IMPLEMENTATION DEFINED features are supported. |
| 0b1100 | IMPLEMENTATION DEFINED features are supported. |
| 0b1101 | IMPLEMENTATION DEFINED features are supported. |
| 0b1110 | IMPLEMENTATION DEFINED features are supported. |
| 0b1111 | IMPLEMENTATION DEFINED features are supported. |

Use of nonzero values requires written permission from Arm.

Access to this field is **RO**.

Accessing TRCIMSPEC0

TRCIMSPEC0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|------------|
| ETE | 0x1C0 | TRCIMSPEC0 |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCIMSPEC<n>, IMP DEF Register <n>, n = 1 - 7

The TRCIMSPEC<n> characteristics are:

Purpose

These registers might return information that is specific to an implementation, or enable features specific to an implementation to be programmed. The product Technical Reference Manual describes these registers.

Configuration

External register TRCIMSPEC<n> bits [31:0] are architecturally mapped to AArch64 System register [TRCIMSPEC<n>\[31:0\]](#).

This register is present only when the trace unit implements this OPTIONAL register and FEAT_ETE is implemented. Otherwise, direct accesses to TRCIMSPEC<n> are RES0.

Attributes

TRCIMSPEC<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------------------------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| IMPLEMENTATION DEFINED | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |

IMPLEMENTATION DEFINED, bits [31:0]

IMPLEMENTATION DEFINED.

This field reads as an IMPLEMENTATION DEFINED value and writes to this field have IMPLEMENTATION DEFINED behavior.

Accessing TRCIMSPEC<n>

TRCIMSPEC<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-------------------|--------------|
| ETE | $0x1C0 + (4 * n)$ | TRCIMSPEC<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

TRCITCTRL, Integration Mode Control Register

The TRCITCTRL characteristics are:

Purpose

A component can use TRCITCTRL to dynamically switch between functional mode and integration mode. In integration mode, topology detection is enabled. After switching to integration mode and performing integration tests or topology detection, reset the system to ensure correct behavior of CoreSight and other connected system components.

For additional information, see the CoreSight Architecture Specification.

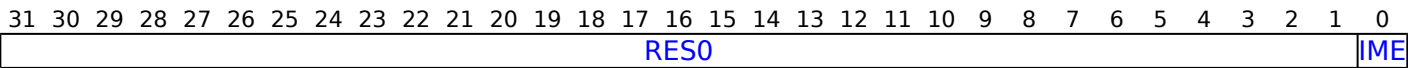
Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCITCTRL are RES0.

Attributes

TRCITCTRL is a 32-bit register.

Field descriptions



Bits [31:1]

Reserved, RES0.

IME, bit [0]

When topology detection or integration functionality is implemented:

Integration Mode Enable.

| IME | Meaning |
|-----|--|
| 0b0 | Component functional mode. |
| 0b1 | Component integration mode. Support for topology detection and integration testing is enabled. |

Otherwise:

Reserved, RES0.

Accessing TRCITCTRL

External debugger accesses to this register are IMPLEMENTATION DEFINED when the trace unit is not in the Idle state.

TRCITCTRL can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| ETE | 0xF00 | TRCITCTRL |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCLAR, Lock Access Register

The TRCLAR characteristics are:

Purpose

Used to lock and unlock the Software Lock.

Note that ETE does not implement the Software Lock.

For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCLAR are RES0.

Attributes

TRCLAR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | KEY | | | | | | | | | | | | | | | |

KEY, bits [31:0]

When Software Lock is implemented:

Software Lock Key.

A value of 0xC5ACCE55 unlocks the Software Lock.

Any other value locks the Software Lock.

Otherwise:

Reserved, RES0.

Accessing TRCLAR

External debugger accesses to this register are unaffected by the OS Lock.

TRCLAR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFB0 | TRCLAR |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **WO**.

TRCLSR, Lock Status Register

The TRCLSR characteristics are:

Purpose

Indicates whether the Software Lock is implemented, and the current status of the Software Lock.
For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCLSR are RES0.

Attributes

TRCLSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-----|----|----|-----|---|-----|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | nTT | | | SLK | | SLI | | | | | | | | |

Bits [31:3]

Reserved, RES0.

nTT, bit [2]

Software lock size.

Reads as 0b0.

Access to this field is **RO**.

SLK, bit [1]

The current Software Lock status.

| SLK | Meaning |
|-----|--|
| 0b0 | Software Lock is unlocked. |
| 0b1 | Software Lock is locked. Writes to the other registers in this component, except for the TRCLAR , are ignored. |

This field reads as 0.

SLI, bit [0]

Indicates whether the Software Lock is implemented.

| SLI | Meaning |
|-----|---|
| 0b0 | Software Lock is not implemented. Writes to the TRCLAR are ignored. |
| 0b1 | Software Lock is implemented. |

This field reads as 0.

Accessing TRCLSR

External debugger accesses to this register are unaffected by the OS Lock.

TRCLSR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFB4 | TRCLSR |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCOSLSR, Trace OS Lock Status Register

The TRCOSLSR characteristics are:

Purpose

Returns the status of the Trace OS Lock.

Configuration

External register TRCOSLSR bits [31:0] are architecturally mapped to AArch64 System register [TRCOSLSR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCOSLSR are RES0.

Attributes

TRCOSLSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|-----------|---|------|---|------|---|---------|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | OSLM[2:1] | | RES0 | | OSLK | | OSLM[0] | |

Bits [31:5]

Reserved, RES0.

OSLM, bits [4:3, 0]

OS Lock model.

| OSLM | Meaning |
|-------|---|
| 0b000 | Trace OS Lock is not implemented. |
| 0b010 | Trace OS Lock is implemented. |
| 0b100 | Trace OS Lock is not implemented, and the trace unit is controlled by the PE OS Lock. |

All other values are reserved.

This field reads as 0b100.

The OSLM field is split as follows:

- OSLM[2:1] is TRCOSLSR[4:3].
- OSLM[0] is TRCOSLSR[0].

Bit [2]

Reserved, RES0.

OSLK, bit [1]

OS Lock status.

| OSLK | Meaning |
|------|--------------------------|
| 0b0 | The OS Lock is unlocked. |
| 0b1 | The OS Lock is locked. |

Note that this field indicates the state of the PE OS Lock.

Accessing TRCOSLSR

External debugger accesses to this register are unaffected by the OS Lock.

TRCOSLSR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x304 | TRCOSLSR |

This interface is accessible as follows:

- When !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPDCR, PowerDown Control Register

The TRCPDCR characteristics are:

Purpose

Requests the system to provide power to the trace unit.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCPDCR are RES0.

Attributes

TRCPDCR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|----|------|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | PU | RES0 | | |

Bits [31:4]

Reserved, RES0.

PU, bit [3]

Power Up Request.

| PU | Meaning |
|-----|---|
| 0b0 | The system can remove power from the trace unit core power domain, or requests for power to the trace unit core power domain are implemented outside of the trace unit. |
| 0b1 | The system must provide power to the trace unit core power domain. |

This field is RES0.

Bits [2:0]

Reserved, RES0.

Accessing TRCPDCR

External debugger accesses to this register are unaffected by the OS Lock.

TRCPDCR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x310 | TRCPDCR |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

TRCPDSR, PowerDown Status Register

The TRCPDSR characteristics are:

Purpose

Indicates the power status of the trace unit.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCPDSR are RES0.

Attributes

TRCPDSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|------|----|------|----|----------|---|-------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | RES0 | | | | | | OSLK | | RES0 | | STICKYPD | | POWER | | | | | | | |

Bits [31:6]

Reserved, RES0.

OSLK, bit [5]

OS Lock Status.

| OSLK | Meaning |
|------|--------------------------|
| 0b0 | The OS Lock is unlocked. |
| 0b1 | The OS Lock is locked. |

Note that this field indicates the state of the PE OS Lock.

Bits [4:2]

Reserved, RES0.

STICKYPD, bit [1]

Sticky powerdown status. Indicates whether the trace register state is valid.

| STICKYPD | Meaning |
|----------|---|
| 0b0 | The state of TRCOSLSR and the trace registers are valid. |
| 0b1 | The state of TRCOSLSR and the trace registers might not be valid. |

This field is set to 1 if the power to the trace unit core power domain is removed and the trace unit register state is not valid.

The STICKYPD field is read-sensitive. On a read of the TRCPDSR, this field is cleared to 0 after the register has been read.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to 1.

POWER, bit [0]

Power Status.

| POWER | Meaning |
|--------------|---|
| 0b0 | The trace unit core power domain is not powered. All trace unit registers are not accessible and they all return an error response. |
| 0b1 | The trace unit core power domain is powered. Trace unit registers are accessible. |

Access to this field is **RAO/WI**.

Accessing TRCPDSR

External debugger accesses to this register are unaffected by the OS Lock.

TRCPDSR can be accessed through the external debug interface:

| Component | Offset | Instance |
|------------------|---------------|-----------------|
| ETE | 0x314 | TRCPDSR |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR0, Peripheral Identification Register 0

The TRCPIDR0 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCPIDR0 are RES0.

Attributes

TRCPIDR0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|--------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | PART_0 | | | | | | | |

Bits [31:8]

Reserved, RES0.

PART_0, bits [7:0]

Part number, bits [7:0].

The part number is selected by the designer of the component, and is stored in [TRCPIDR1](#).PART_1 and TRCPIDR0.PART_0.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCPIDR0

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR0 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFE0 | TRCPIDR0 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

TRCPIDR1, Peripheral Identification Register 1

The TRCPIDR1 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCPIDR1 are RES0.

Attributes

TRCPIDR1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|-------|---|---|--------|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | DES 0 | | | PART 1 | | | | |

Bits [31:8]

Reserved, RES0.

DES_0, bits [7:4]

Designer, JEP106 identification code, bits [3:0]. TRCPIDR1.DES_0 and [TRCPIDR2.DES_1](#) together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC <http://www.jedec.org>.

Note

For a component designed by Arm Limited, the JEP106 identification code is 0x3B.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

PART_1, bits [3:0]

Part number, bits [11:8].

The part number is selected by the designer of the component, and is stored in TRCPIDR1.PART_1 and [TRCPIDR0.PART_0](#).

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCPIDR1

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFE4 | TRCPIDR1 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR2, Peripheral Identification Register 2

The TRCPIDR2 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCPIDR2 are RES0.

Attributes

TRCPIDR2 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|----------|---|---|---|-------|---|-------|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | REVISION | | | | JEDEC | | DES 1 | |

Bits [31:8]

Reserved, RES0.

REVISION, bits [7:4]

Component major revision. TRCPIDR2.REVISION and [TRCPIDR3.REVAND](#) together form the revision number of the component, with TRCPIDR2.REVISION being the most significant part and [TRCPIDR3.REVAND](#) the least significant part. When a component is changed, TRCPIDR2.REVISION or [TRCPIDR3.REVAND](#) are increased to ensure that software can differentiate the different revisions of the component. [TRCPIDR3.REVAND](#) should be set to 0b0000 when TRCPIDR2.REVISION is increased.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

JEDEC, bit [3]

JEDEC-assigned JEP106 implementer code is used.

Reads as 0b1.

Access to this field is **RO**.

DES_1, bits [2:0]

Designer, JEP106 identification code, bits [6:4]. [TRCPIDR1.DES_0](#) and TRCPIDR2.DES_1 together form the JEDEC-assigned JEP106 identification code for the designer of the component. The parity bit in the JEP106 identification code is not included. The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC <http://www.jedec.org>.

Note

For a component designed by Arm Limited, the JEP106 identification code is 0x3B.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCPIDR2

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR2 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFE8 | TRCPIDR2 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR3, Peripheral Identification Register 3

The TRCPIDR3 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCPIDR3 are RES0.

Attributes

TRCPIDR3 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|--------|---|---|---|------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | REVAND | | | | CMOD | | | |

Bits [31:8]

Reserved, RES0.

REVAND, bits [7:4]

Component minor revision. [TRCPIDR2.REVISION](#) and TRCPIDR3.REVAND together form the revision number of the component, with [TRCPIDR2.REVISION](#) being the most significant part and TRCPIDR3.REVAND the least significant part. When a component is changed, [TRCPIDR2.REVISION](#) or TRCPIDR3.REVAND are increased to ensure that software can differentiate the different revisions of the component. TRCPIDR3.REVAND should be set to 0b0000 when [TRCPIDR2.REVISION](#) is increased.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

CMOD, bits [3:0]

Customer Modified.

Indicates the component has been modified.

A value of 0b0000 means the component is not modified from the original design.

Any other value means the component has been modified in an IMPLEMENTATION DEFINED way.

For any two components with the same Unique Component Identifier:

- If the value of the CMOD fields of both components equals zero, the components are identical.
- If the CMOD fields of both components have the same non-zero value, it does not necessarily mean that they have the same modifications.
- If the value of the CMOD field of either of the two components is non-zero, they might not be identical, even though they have the same Unique Component Identifier.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCPIDR3

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR3 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFEC | TRCPIDR3 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR4, Peripheral Identification Register 4

The TRCPIDR4 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCPIDR4 are RES0.

Attributes

TRCPIDR4 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|----|----|----|----|---|---|---|---|---|------|---|---|-------|---|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | SIZE | | | DES 2 | | |

Bits [31:8]

Reserved, RES0.

SIZE, bits [7:4]

Size of the component.

The distance from the start of the address space used by this component to the end of the component identification registers.

A value of 0b0000 means one of the following is true:

- The component uses a single 4KB block.
- The component uses an IMPLEMENTATION DEFINED number of 4KB blocks.

Any other value means the component occupies $2^{\text{TRCPIDR4.SIZE}}$ 4KB blocks.

Using this field to indicate the size of the component is deprecated. This field might not correctly indicate the size of the component. Arm recommends that software determine the size of the component from the Unique Component Identifier fields, and other IMPLEMENTATION DEFINED registers in the component.

Reads as 0b0000.

Access to this field is **RO**.

DES_2, bits [3:0]

Designer, JEP106 continuation code. This is the JEDEC-assigned JEP106 bank identifier for the designer of the component, minus 1. The code identifies the designer of the component, which might not be the same as the implementer of the device containing the component. To obtain a number, or to see the assignment of these codes, contact JEDEC <http://www.jedec.org>.

Note

For a component designed by Arm Limited, the JEP106 bank is 5, meaning this field has the value 0x4.

This field has an IMPLEMENTATION DEFINED value.

Access to this field is **RO**.

Accessing TRCPIDR4

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR4 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFD0 | TRCPIDR4 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR5, Peripheral Identification Register 5

The TRCPIDR5 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCPIDR5 are RES0.

Attributes

TRCPIDR5 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |

Bits [31:0]

Reserved, RES0.

Accessing TRCPIDR5

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR5 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFD4 | TRCPIDR5 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR6, Peripheral Identification Register 6

The TRCPIDR6 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCPIDR6 are RES0.

Attributes

TRCPIDR6 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |

Bits [31:0]

Reserved, RES0.

Accessing TRCPIDR6

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR6 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFD8 | TRCPIDR6 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPIDR7, Peripheral Identification Register 7

The TRCPIDR7 characteristics are:

Purpose

Provides discovery information about the component.

For additional information, see the CoreSight Architecture Specification.

Configuration

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCPIDR7 are RES0.

Attributes

TRCPIDR7 is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| | | | | | | | | | | | | | | | | RES0 | | | | | | | | | | | | | | | |

Bits [31:0]

Reserved, RES0.

Accessing TRCPIDR7

External debugger accesses to this register are unaffected by the OS Lock.

TRCPIDR7 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0xFDC | TRCPIDR7 |

This interface is accessible as follows:

- When !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCPRGCTLR, Programming Control Register

The TRCPRGCTLR characteristics are:

Purpose

Enables the trace unit.

Configuration

External register TRCPRGCTLR bits [31:0] are architecturally mapped to AArch64 System register [TRCPRGCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCPRGCTLR are RES0.

Attributes

TRCPRGCTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|----|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | EN |

Bits [31:1]

Reserved, RES0.

EN, bit [0]

Trace unit enable.

| EN | Meaning |
|-----|-----------------------------|
| 0b0 | The trace unit is disabled. |
| 0b1 | The trace unit is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to 0.

Accessing TRCPRGCTLR

Must be programmed.

TRCPRGCTLR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|------------|
| ETE | 0x004 | TRCPRGCTLR |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

TRCQCTLR, Q Element Control Register

The TRCQCTLR characteristics are:

Purpose

Controls when Q elements are enabled.

Configuration

External register TRCQCTLR bits [31:0] are architecturally mapped to AArch64 System register [TRCQCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR0.QFILT == 1. Otherwise, direct accesses to TRCQCTLR are RES0.

Attributes

TRCQCTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|------|----------|----------|----------|----------|----------|----------|----------|----------|----|----|----|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 |
| RES0 | | | | | | | | | | MODE | RANGE[7] | RANGE[6] | RANGE[5] | RANGE[4] | RANGE[3] | RANGE[2] | RANGE[1] | RANGE[0] | | | | | | | | | | | |

Bits [31:9]

Reserved, RES0.

MODE, bit [8]

Selects whether the Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit is permitted to generate Q elements or address ranges where the trace unit is not permitted to generate Q elements:

| MODE | Meaning |
|------|--|
| 0b0 | Exclude mode. The Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit must not generate Q elements. If no ranges are selected, Q elements are permitted across the entire memory map. |
| 0b1 | Include Mode. The Address Range Comparators selected by TRCQCTLR.RANGE indicate address ranges where the trace unit can generate Q elements. If all the implemented bits in RANGE are set to 0 then Q elements are disabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

RANGE[<m>], bit [m], for m = 7 to 0

Specifies whether Address Range Comparator <m> controls Q elements.

| RANGE[<m>] | Meaning |
|-------------------------|---|
| 0b0 | The address range that Address Range Comparator <m> defines, is not selected. |
| 0b1 | The address range that Address Range Comparator <m> defines, is selected. |

This bit is RES0 if m >= [TRCIDR4](#).NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCQCTLR

Must be programmed if [TRCCONFIGR](#).QE != 0b00.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCQCTLR can be accessed through the external debug interface:

| Component | Offset | Instance |
|------------------|---------------|-----------------|
| ETE | 0x044 | TRCQCTLR |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCRSCTLR<n>, Resource Selection Control Register <n>, n = 2 - 31

The TRCRSCTLR<n> characteristics are:

Purpose

Controls the selection of the resources in the trace unit.

Configuration

External register TRCRSCTLR<n> bits [31:0] are architecturally mapped to AArch64 System register [TRCRSCTLR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and $((\text{TRCIDR4.NUMRSPAIR} + 1) * 2) > n$. Otherwise, direct accesses to TRCRSCTLR<n> are RES0.

Resource selector 0 always returns FALSE.

Resource selector 1 always returns TRUE.

Resource selectors are implemented in pairs. Each odd numbered resource selector is part of a pair with the even numbered resource selector that is numbered as one less than it. For example, resource selectors 2 and 3 form a pair.

Attributes

TRCRSCTLR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|---------|-----|-------|----|----|----|--------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | PAIRINV | INV | GROUP | | | | SELECT | | | | | | | | | | | | | | | |

Bits [31:22]

Reserved, RES0.

PAIRINV, bit [21]

When n is even:

Controls whether the combined result from a resource selector pair is inverted.

| PAIRINV | Meaning |
|---------|--|
| 0b0 | Do not invert the combined output of the 2 resource selectors. |
| 0b1 | Invert the combined output of the 2 resource selectors. |

If:

- A is the register TRCRSCTLR<n>.
- B is the register TRCRSCTLR<n+1>.

Then the combined output of the 2 resource selectors A and B depends on the value of (A.PAIRINV, A.INV, B.INV) as follows:

- 0b000 -> A and B.
- 0b001 -> Reserved.
- 0b010 -> not(A) and B.
- 0b011 -> not(A) and not(B).

- 0b100 -> not(A) or not(B).
- 0b101 -> not(A) or B.
- 0b110 -> Reserved.
- 0b111 -> A or B.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

INV, bit [20]

Controls whether the resource, that TRCRSCTLR<n>.GROUP and TRCRSCTLR<n>.SELECT selects, is inverted.

| INV | Meaning |
|-----|--|
| 0b0 | Do not invert the output of this selector. |
| 0b1 | Invert the output of this selector. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

GROUP, bits [19:16]

Selects a group of resources.

| GROUP | Meaning | SELECT |
|--------|---|--|
| 0b0000 | External Input Selectors. | SELECT encoding for External Input Selectors |
| 0b0001 | PE Comparator Inputs. | SELECT encoding for PE Comparator Inputs |
| 0b0010 | Counters and Sequencer. | SELECT encoding for Counters and Sequencer |
| 0b0011 | Single-shot Comparator Controls. | SELECT encoding for Single-shot Comparator Controls |
| 0b0100 | Single Address Comparators. | SELECT encoding for Single Address Comparators |
| 0b0101 | Address Range Comparators. | SELECT encoding for Address Range Comparators |
| 0b0110 | Context Identifier Comparators. | SELECT encoding for Context Identifier Comparators |
| 0b0111 | Virtual Context Identifier Comparators. | SELECT encoding for Virtual Context Identifier Comparators |

All other values are reserved.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT, bits [15:0]

Resource Specific Controls. Contains the controls specific to the resource group selected by GROUP, described in the following sections.

SELECT encoding for External Input Selectors

| | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|---|---|---|---|---|---|----------|----------|----------|----------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | EXTIN[3] | EXTIN[2] | EXTIN[1] | EXTIN[0] |

Bits [15:4]

Reserved, RES0.

EXTIN[<m>], bit [m], for m = 3 to 0

Selects one or more External Inputs.

| EXTIN[<m>] | Meaning |
|------------|-------------------|
| 0b0 | Ignore EXTIN <m>. |
| 0b1 | Select EXTIN <m>. |

This bit is RES0 if m >= [TRCIDR5](#).NUMEXTINSEL.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for PE Comparator Inputs

| | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|-----------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | |
| | | | | | | | | RES0 | PECOMP[7] | PECOMP[6] | PECOMP[5] | PECOMP[4] | PECOMP[3] | PECOMP[2] | PECOMP[1] | PECOMP[0] |

Bits [15:8]

Reserved, RES0.

PECOMP[<m>], bit [m], for m = 7 to 0

Selects one or more PE Comparator Inputs.

| PECOMP[<m>] | Meaning |
|-------------|---------------------------------|
| 0b0 | Ignore PE Comparator Input <m>. |
| 0b1 | Select PE Comparator Input <m>. |

This bit is RES0 if m >= [TRCIDR4](#).NUMPC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Counters and Sequencer

| | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|---|---|------|--------------|--------------|--------------|--------------|-------------|-------------|-------------|-------------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | | | |
| | | | | | | | | RES0 | SEQUENCER[3] | SEQUENCER[2] | SEQUENCER[1] | SEQUENCER[0] | COUNTERS[3] | COUNTERS[2] | COUNTERS[1] | COUNTERS[0] |

Bits [15:8]

Reserved, RES0.

SEQUENCER[<m>], bit [m+4], for m = 3 to 0

Sequencer states.

| SEQUENCER[<m>] | Meaning |
|----------------|-----------------------------|
| 0b0 | Ignore Sequencer state <m>. |
| 0b1 | Select Sequencer state <m>. |

This bit is RES0 if m >= [TRCIDR5](#).NUMSEQSTATE.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

COUNTERS[<m>], bit [m], for m = 3 to 0

Counters resources at zero.

| COUNTERS[<m>] | Meaning |
|---------------|-----------------------------|
| 0b0 | Ignore Counter <m>. |
| 0b1 | Select Counter <m> is zero. |

This bit is RES0 if m >= [TRCIDR5](#).NUMCNTR.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Single-shot Comparator Controls

| | | | | | | | | | | | | | | | |
|------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|----------------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | SINGLE_SHOT[7] | SINGLE_SHOT[6] | SINGLE_SHOT[5] | SINGLE_SHOT[4] | SINGLE_SHOT[3] | SINGLE_SHOT[2] | SINGLE_SHOT[1] | SINGLE_SHOT[0] | SINGLE_SHOT[7] | SINGLE_SHOT[6] | SINGLE_SHOT[5] | SINGLE_SHOT[4] | SINGLE_SHOT[3] | SINGLE_SHOT[2] | SINGLE_SHOT[1] |

Bits [15:8]

Reserved, RES0.

SINGLE_SHOT[<m>], bit [m], for m = 7 to 0

Selects one or more Single-shot Comparator Controls.

| SINGLE_SHOT[<m>] | Meaning |
|------------------|--|
| 0b0 | Ignore Single-shot Comparator Control <m>. |
| 0b1 | Select Single-shot Comparator Control <m>. |

This bit is RES0 if m >= [TRCIDR4](#).NUMSSCC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Single Address Comparators

| | | | | | | | | | | | | | | | |
|---------|---------|---------|---------|---------|---------|--------|--------|--------|--------|--------|--------|--------|--------|--------|--------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| SAC[15] | SAC[14] | SAC[13] | SAC[12] | SAC[11] | SAC[10] | SAC[9] | SAC[8] | SAC[7] | SAC[6] | SAC[5] | SAC[4] | SAC[3] | SAC[2] | SAC[1] | SAC[0] |

SAC[<m>], bit [m], for m = 15 to 0

Selects one or more Single Address Comparators.

| SAC[<m>] | Meaning |
|----------|---------------------------------------|
| 0b0 | Ignore Single Address Comparator <m>. |
| 0b1 | Select Single Address Comparator <m>. |

This bit is RES0 if m >= 2 × [TRCIDR4](#).NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Address Range Comparators

| | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|---|---|--------|--------|--------|--------|--------|--------|--------|--------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | ARC[7] | ARC[6] | ARC[5] | ARC[4] | ARC[3] | ARC[2] | ARC[1] | ARC[0] |

Bits [15:8]

Reserved, RES0.

ARC[<m>], bit [m], for m = 7 to 0

Selects one or more Address Range Comparators.

| ARC[<m>] | Meaning |
|----------|--------------------------------------|
| 0b0 | Ignore Address Range Comparator <m>. |
| 0b1 | Select Address Range Comparator <m>. |

This bit is RES0 if m >= [TRCIDR4](#).NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Context Identifier Comparators

| | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|---|---|--------|--------|--------|--------|--------|--------|--------|--------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | CID[7] | CID[6] | CID[5] | CID[4] | CID[3] | CID[2] | CID[1] | CID[0] |

Bits [15:8]

Reserved, RES0.

CID[<m>], bit [m], for m = 7 to 0

Selects one or more Context Identifier Comparators.

| CID[<m>] | Meaning |
|----------|---|
| 0b0 | Ignore Context Identifier Comparator <m>. |
| 0b1 | Select Context Identifier Comparator <m>. |

This bit is RES0 if m >= [TRCIDR4](#).NUMCIDC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SELECT encoding for Virtual Context Identifier Comparators

| | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|---|---|---------|---------|---------|---------|---------|---------|---------|---------|
| 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | VMID[7] | VMID[6] | VMID[5] | VMID[4] | VMID[3] | VMID[2] | VMID[1] | VMID[0] |

Bits [15:8]

Reserved, RES0.

VMID[<m>], bit [m], for m = 7 to 0

Selects one or more Virtual Context Identifier Comparators.

| VMID[<m>] | Meaning |
|-----------|---|
| 0b0 | Ignore Virtual Context Identifier Comparator <m>. |
| 0b1 | Select Virtual Context Identifier Comparator <m>. |

This bit is RES0 if m >= [TRCIDR4.NUMVMIDC](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCRSCTLR<n>

Must be programmed if any of the following are true:

- [TRCCNTCTLR<a>.RLDEVENT](#).TYPE == 0 and [TRCCNTCTLR<a>.RLDEVENT](#).SEL == n.
- [TRCCNTCTLR<a>.RLDEVENT](#).TYPE == 1 and [TRCCNTCTLR<a>.RLDEVENT](#).SEL == n/2.
- [TRCCNTCTLR<a>.CNTEVENT](#).TYPE == 0 and [TRCCNTCTLR<a>.CNTEVENT](#).SEL == n.
- [TRCCNTCTLR<a>.CNTEVENT](#).TYPE == 1 and [TRCCNTCTLR<a>.CNTEVENT](#).SEL == n/2.
- [TRCEVENTCTL0R.EVENT0](#).TYPE == 0 and [TRCEVENTCTL0R.EVENT0](#).SEL == n.
- [TRCEVENTCTL0R.EVENT0](#).TYPE == 1 and [TRCEVENTCTL0R.EVENT0](#).SEL == n/2.
- [TRCEVENTCTL0R.EVENT1](#).TYPE == 0 and [TRCEVENTCTL0R.EVENT1](#).SEL == n.
- [TRCEVENTCTL0R.EVENT1](#).TYPE == 1 and [TRCEVENTCTL0R.EVENT1](#).SEL == n/2.
- [TRCEVENTCTL0R.EVENT2](#).TYPE == 0 and [TRCEVENTCTL0R.EVENT2](#).SEL == n.
- [TRCEVENTCTL0R.EVENT2](#).TYPE == 1 and [TRCEVENTCTL0R.EVENT2](#).SEL == n/2.
- [TRCEVENTCTL0R.EVENT3](#).TYPE == 0 and [TRCEVENTCTL0R.EVENT3](#).SEL == n.
- [TRCEVENTCTL0R.EVENT3](#).TYPE == 1 and [TRCEVENTCTL0R.EVENT3](#).SEL == n/2.
- [TRCSEQEVR<a>.B](#).TYPE == 0 and [TRCSEQEVR<a>.B](#).SEL == n.
- [TRCSEQEVR<a>.B](#).TYPE == 1 and [TRCSEQEVR<a>.B](#).SEL == n/2.
- [TRCSEQEVR<a>.F](#).TYPE == 0 and [TRCSEQEVR<a>.F](#).SEL == n.
- [TRCSEQEVR<a>.F](#).TYPE == 1 and [TRCSEQEVR<a>.F](#).SEL == n/2.
- [TRCSEQRSTEVR.RST](#).TYPE == 0 and [TRCSEQRSTEVR.RST](#).SEL == n.
- [TRCSEQRSTEVR.RST](#).TYPE == 1 and [TRCSEQRSTEVR.RST](#).SEL == n/2.
- [TRCTSCTLR.EVENT](#).TYPE == 0 and [TRCTSCTLR.EVENT](#).SEL == n.
- [TRCTSCTLR.EVENT](#).TYPE == 1 and [TRCTSCTLR.EVENT](#).SEL == n/2.
- [TRCVICTLR.EVENT](#).TYPE == 0 and [TRCVICTLR.EVENT](#).SEL == n.
- [TRCVICTLR.EVENT](#).TYPE == 1 and [TRCVICTLR.EVENT](#).SEL == n/2.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCRSCTLR<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|--------------|
| ETE | 0x200 + (4 * n) | TRCRSCTLR<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

TRCRSR, Resources Status Register

The TRCRSR characteristics are:

Purpose

Use this to set, or read, the status of the resources.

Configuration

External register TRCRSR bits [31:0] are architecturally mapped to AArch64 System register [TRCRSR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCRSR are RES0.

Attributes

TRCRSR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----------|----------|----------|----------|------|----------|----------|----------|----------|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | TA | EVENT[3] | EVENT[2] | EVENT[1] | EVENT[0] | RES0 | EXTIN[3] | EXTIN[2] | EXTIN[1] | EXTIN[0] | | | | | | | | | | |

Bits [31:13]

Reserved, RES0.

TA, bit [12]

Tracing active.

| TA | Meaning |
|-----|------------------------|
| 0b0 | Tracing is not active. |
| 0b1 | Tracing is active. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

EVENT[<m>], bit [m+8], for m = 3 to 0

Untraced status of ETEEvents.

| EVENT[<m>] | Meaning |
|------------|--|
| 0b0 | An ETEEvent <m> has not occurred. |
| 0b1 | An ETEEvent <m> has occurred while the resources were in the Paused state. |

This bit is RES0 if [TRCIDR4](#).NUMRSPAIR == 0 || m > [TRCIDR0](#).NUMEVENT.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

EXTIN[<m>], bit [m], for m = 3 to 0

The sticky status of the External Input Selectors.

| EXTIN[<m>] | Meaning |
|-------------------------|---|
| 0b0 | An event selected by External Input Selector <m> has not occurred. |
| 0b1 | At least one event selected by External Input Selector <m> has occurred while the resources were in the Paused state. |

This bit is RES0 if m >= [TRCIDR5.NUMEXTINSEL](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCRSR

Must always be programmed.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCRSR can be accessed through the external debug interface:

| Component | Offset | Instance |
|------------------|---------------|-----------------|
| ETE | 0x028 | TRCRSR |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSEQEVR<n>, Sequencer State Transition Control Register <n>, n = 0 - 2

The TRCSEQEVR<n> characteristics are:

Purpose

Moves the Sequencer state:

- Backwards, from state n+1 to state n when a programmed resource event occurs.
- Forwards, from state n to state n+1 when a programmed resource event occurs.

Configuration

External register TRCSEQEVR<n> bits [31:0] are architecturally mapped to AArch64 System register [TRCSEQEVR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQEVR<n> are RES0.

Attributes

TRCSEQEVR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|------|----|-------|----|---|---|--------|---|------|---|-------|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | B_TYPE | | RES0 | | B_SEL | | | | F_TYPE | | RES0 | | F_SEL | | | |

Bits [31:16]

Reserved, RES0.

B_TYPE, bit [15]

Chooses the type of Resource Selector.

Backward field. Defines whether the backward resource event is a single Resource Selector or a Resource Selector pair. When the resource event occurs then the Sequencer state moves from state n+1 to state n. For example, if TRCSEQEVR2.B.SEL == 0x14 then when event 0x14 occurs, the Sequencer moves from state 3 to state 2.

| B_TYPE | Meaning |
|--------|---|
| 0b0 | A single Resource Selector. TRCSEQEVR<n>.B.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCSEQEVR<n>.B.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR<n>.B.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [14:13]

Reserved, RES0.

B_SEL, bits [12:8]

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR<n>.B.TYPE controls whether TRCSEQEVR<n>.B.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Backward field. Selects the single Resource Selector or Resource Selector pair.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

F_TYPE, bit [7]

Chooses the type of Resource Selector.

Backward field. Defines whether the forward resource event is a single Resource Selector or a Resource Selector pair. When the resource event occurs then the Sequencer state moves from state n to state n+1. For example, if TRCSEQEVR1.F.SEL == 0x12 then when event 0x12 occurs, the Sequencer moves from state 1 to state 2.

| F_TYPE | Meaning |
|---------------|---|
| 0b0 | A single Resource Selector. TRCSEQEVR<n>.F.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCSEQEVR<n>.F.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQEVR<n>.F.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

F_SEL, bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQEVR<n>.F.TYPE controls whether TRCSEQEVR<n>.F.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

Forward field. Selects the single Resource Selector or Resource Selector pair.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQEVR<n>

Must be programmed if [TRCRSCTLR<a>.GROUP == 0b0010](#) and [TRCRSCTLR<a>.SEQUENCER != 0b0000](#).

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCSEQEVR<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-------------------|--------------|
| ETE | $0x100 + (4 * n)$ | TRCSEQEVR<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSEQRSTEVR, Sequencer Reset Control Register

The TRCSEQRSTEVR characteristics are:

Purpose

Moves the Sequencer to state 0 when a programmed resource event occurs.

Configuration

External register TRCSEQRSTEVR bits [31:0] are architecturally mapped to AArch64 System register [TRCSEQRSTEVR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQRSTEVR are RES0.

Attributes

TRCSEQRSTEVR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----------|----|------|----|---------|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | RST TYPE | | RES0 | | RST SEL | | | | | | | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

RST_TYPE, bit [7]

Chooses the type of Resource Selector.

| RST_TYPE | Meaning |
|----------|--|
| 0b0 | A single Resource Selector. TRCSEQRSTEVR.RST.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCSEQRSTEVR.RST.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCSEQRSTEVR.RST.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [6:5]

Reserved, RES0.

RST_SEL, bits [4:0]

Defines the selected Resource Selector or pair of Resource Selectors. TRCSEQRSTEVR.RST.TYPE controls whether TRCSEQRSTEVR.RST.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQRSTEV

Must be programmed if [TRCRSCTLR<a>.GROUP == 0b0010](#) and [TRCRSCTLR<a>.SEQUENCER != 0b0000](#).

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCSEQRSTEV can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-------------|
| ETE | 0x118 | TRCSEQRSTEV |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSEQSTR, Sequencer State Register

The TRCSEQSTR characteristics are:

Purpose

Use this to set, or read, the Sequencer state.

Configuration

External register TRCSEQSTR bits [31:0] are architecturally mapped to AArch64 System register [TRCSEQSTR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR5.NUMSEQSTATE != 0b000. Otherwise, direct accesses to TRCSEQSTR are RES0.

Attributes

TRCSEQSTR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|-------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | STATE |

Bits [31:2]

Reserved, RES0.

STATE, bits [1:0]

Set or returns the state of the Sequencer.

| STATE | Meaning |
|-------|----------|
| 0b00 | State 0. |
| 0b01 | State 1. |
| 0b10 | State 2. |
| 0b11 | State 3. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSEQSTR

Must be programmed if [TRCRSCTLR<a>.GROUP](#) == 0b0010 and [TRCRSCTLR<a>.SEQUENCER](#) != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCSEQSTR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| ETE | 0x11C | TRCSEQSTR |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSSCCR<n>, Single-shot Comparator Control Register <n>, n = 0 - 7

The TRCSSCCR<n> characteristics are:

Purpose

Controls the corresponding Single-shot Comparator Control resource.

Configuration

External register TRCSSCCR<n> bits [31:0] are architecturally mapped to AArch64 System register [TRCSSCCR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMSSCC > n. Otherwise, direct accesses to TRCSSCCR<n> are RES0.

Attributes

TRCSSCCR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|------|-----|--------|--------|--------|--------|--------|--------|--------|--------|---------|---------|---------|---------|---------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | |
| | | | | | | | RES0 | RST | ARC[7] | ARC[6] | ARC[5] | ARC[4] | ARC[3] | ARC[2] | ARC[1] | ARC[0] | SAC[15] | SAC[14] | SAC[13] | SAC[12] | SAC[11] |

Bits [31:25]

Reserved, RES0.

RST, bit [24]

Selects the Single-shot Comparator Control mode.

| RST | Meaning |
|-----|--|
| 0b0 | The Single-shot Comparator Control is in single-shot mode. |
| 0b1 | The Single-shot Comparator Control is in multi-shot mode. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

ARC[<m>], bit [m+16], for m = 7 to 0

Selects one or more Address Range Comparators for Single-shot control.

| ARC[<m>] | Meaning |
|----------|--|
| 0b0 | The Address Range Comparator <m>, is not selected for Single-shot control. |
| 0b1 | The Address Range Comparator <m>, is selected for Single-shot control. |

This bit is RES0 if m >= [TRCIDR4.NUMACPAIRS](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SAC[<m>], bit [m], for m = 15 to 0

Selects one or more Single Address Comparators for Single-shot control.

| SAC[<m>] | Meaning |
|-----------------------|---|
| 0b0 | The Single Address Comparator <m>, is not selected for Single-shot control. |
| 0b1 | The Single Address Comparator <m>, is selected for Single-shot control. |

This bit is RES0 if $m \geq 2 \times \text{TRCIDR4.NUMACPAIRS}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSSCCR<n>

Must be programmed if any [TRCRSCTLR<a>](#).GROUP == 0b0011 and [TRCRSCTLR<a>](#).SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCSSCCR<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|------------------|-------------------|-----------------|
| ETE | $0x280 + (4 * n)$ | TRCSSCCR<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSSCSR<n>, Single-shot Comparator Control Status Register <n>, n = 0 - 7

The TRCSSCSR<n> characteristics are:

Purpose

Returns the status of the corresponding Single-shot Comparator Control.

Configuration

External register TRCSSCSR<n> bits [31:0] are architecturally mapped to AArch64 System register [TRCSSCSR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMSSCC > n. Otherwise, direct accesses to TRCSSCSR<n> are RES0.

Attributes

TRCSSCSR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|--------|----|---------|----|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|-----|---|-----|---|------|--|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 | | |
| STATUS | | PENDING | | RES0 | | | | | | | | | | | | | | | | | | | | | | | | PCD | | VDA | | INST | |

STATUS, bit [31]

Single-shot Comparator Control status. Indicates if any of the comparators selected by this Single-shot Comparator control have matched. The selected comparators are defined by [TRCSSCCR<n>.ARC](#), [TRCSSCCR<n>.SAC](#), and [TRCSSPCICR<n>.PC](#).

| STATUS | Meaning |
|--------|---|
| 0b0 | No match has occurred. When the first match occurs, this field takes a value of 1. It remains at 1 until explicitly modified by a write to this register. |
| 0b1 | One or more matches has occurred. If TRCSSCCR<n>.RST == 0 then: <ul style="list-style-type: none"> There is only one match and no more matches are possible. Software must reset this field to 0 to re-enable the Single-shot Comparator Control. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

PENDING, bit [30]

Single-shot pending status. The Single-shot Comparator Control fired while the resources were in the Paused state.

| PENDING | Meaning |
|---------|-----------------------------------|
| 0b0 | No match has occurred. |
| 0b1 | One or more matches has occurred. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [29:4]

Reserved, RES0.

PC, bit [3]

PE Comparator Input support. Indicates if the Single-shot Comparator Control supports PE Comparator Inputs.

| PC | Meaning |
|-----|--|
| 0b0 | This Single-shot Comparator Control does not support PE Comparator Inputs. Selecting any PE Comparator Inputs using the associated TRCSSPCICR<n> results in CONSTRAINED UNPREDICTABLE behavior of the Single-shot Comparator Control resource. The Single-shot Comparator Control might match unexpectedly or might not match. |
| 0b1 | This Single-shot Comparator Control supports PE Comparator Inputs. |

Access to this field is **RO**.

DV, bit [2]

Data value comparator support. Data value comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

| DV | Meaning |
|-----|--|
| 0b0 | This Single-shot Comparator Control does not support data value comparisons. |
| 0b1 | This Single-shot Comparator Control supports data value comparisons. |

This field reads as 0.

Access to this field is **RO**.

DA, bit [1]

Data Address Comparator support. Data address comparisons are not implemented in ETE and are reserved for other trace architectures. Allocated in other trace architectures.

| DA | Meaning |
|-----|--|
| 0b0 | This Single-shot Comparator Control does not support data address comparisons. |
| 0b1 | This Single-shot Comparator Control supports data address comparisons. |

This field reads as 0.

Access to this field is **RO**.

INST, bit [0]

Instruction Address Comparator support. Indicates if the Single-shot Comparator Control supports instruction address comparisons.

| INST | Meaning |
|------|---|
| 0b0 | This Single-shot Comparator Control does not support instruction address comparisons. |
| 0b1 | This Single-shot Comparator Control supports instruction address comparisons. |

This field reads as 1.

Access to this field is **RO**.

Accessing TRCSSCSR<n>

Must be programmed if [TRCRSCTLR<a>](#).GROUP == 0b0011 and [TRCRSCTLR<a>](#).SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCSSCSR<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|-------------|
| ETE | 0x2A0 + (4 * n) | TRCSSCSR<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSSPCICR<n>, Single-shot Processing Element Comparator Input Control Register <n>, n = 0 - 7

The TRCSSPCICR<n> characteristics are:

Purpose

Returns the status of the corresponding Single-shot Comparator Control.

Configuration

External register TRCSSPCICR<n> bits [31:0] are architecturally mapped to AArch64 System register [TRCSSPCICR<n>\[31:0\]](#).

This register is present only when FEAT_ETE is implemented, TRCIDR4.NUMSSCC > n, TRCIDR4.NUMPC > 0b0000 and TRCSSCSR<n>.PC == 1. Otherwise, direct accesses to TRCSSPCICR<n> are RES0.

Attributes

TRCSSPCICR<n> is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|--|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | PC[7]PC[6]PC[5]PC[4]PC[3]PC[2]PC[1]PC[0] | | | | | | | | | | | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

PC[<m>], bit [m], for m = 7 to 0

Selects one or more PE Comparator Inputs for Single-shot control.

| PC[<m>] | Meaning |
|---------|---|
| 0b0 | The single PE Comparator Input <m>, is not selected as for Single-shot control. |
| 0b1 | The single PE Comparator Input <m>, is selected as for Single-shot control. |

This bit is RES0 if m >= [TRCIDR4](#).NUMPC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSSPCICR<n>

Must be programmed if implemented and any [TRCRSCTLR<a>](#).GROUP == 0b0011 and [TRCRSCTLR<a>](#).SINGLE_SHOT[n] == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCSSPCICR<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-------------------|---------------|
| ETE | $0x2C0 + (4 * n)$ | TRCSSPCICR<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSTALLCTLR, Stall Control Register

The TRCSTALLCTLR characteristics are:

Purpose

Enables trace unit functionality that prevents trace unit buffer overflows.

Configuration

External register TRCSTALLCTLR bits [31:0] are architecturally mapped to AArch64 System register [TRCSTALLCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR3.STALLCTL == 1. Otherwise, direct accesses to TRCSTALLCTLR are RES0.

Attributes

TRCSTALLCTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|------------|----|------|----|--------|----|------|----|-------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | NOOVERFLOW | | RES0 | | ISTALL | | RES0 | | LEVEL | | | | | | | | | |

Bits [31:14]

Reserved, RES0.

NOOVERFLOW, bit [13]

When TRCIDR3.NOOVERFLOW == 1:

Trace overflow prevention.

| NOOVERFLOW | Meaning |
|------------|--|
| 0b0 | Trace unit buffer overflow prevention is disabled. |
| 0b1 | Trace unit buffer overflow prevention is enabled. |

Note that enabling this feature might cause a significant performance impact.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [12:9]

Reserved, RES0.

ISTALL, bit [8]

Instruction stall control. Controls if a trace unit can stall the PE when the trace buffer space is less than LEVEL.

| ISTALL | Meaning |
|--------|---------------------------------------|
| 0b0 | The trace unit must not stall the PE. |
| 0b1 | The trace unit can stall the PE. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [7:4]

Reserved, RES0.

LEVEL, bits [3:0]

Threshold level field. The field can support 16 monotonic levels from 0b0000 to 0b1111.

The value 0b0000 defines the Minimal invasion level. This setting has a greater risk of a trace unit buffer overflow.

The value 0b1111 defines the Maximum invasion level. This setting has a reduced risk of a trace unit buffer overflow.

Note that for some implementations, invasion might occur at the minimal invasion level.

One or more of the least significant bits of LEVEL are permitted to be RES0. Arm recommends that LEVEL[3:2] are fully implemented. Arm strongly recommends that LEVEL[3] is always implemented. If one or more bits are RES0 and are written with a non-zero value, the effective value of LEVEL is rounded down to the nearest power of 2 value which has the RES0 bits as zero. For example, if LEVEL[1:0] are RES0 and a value of 0b1110 is written to LEVEL, the effective value of LEVEL is 0b1100.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSTALLCTLR

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCSTALLCTLR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|--------------|
| ETE | 0x02C | TRCSTALLCTLR |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

TRCSTATR, Trace Status Register

The TRCSTATR characteristics are:

Purpose

Returns the trace unit status.

Configuration

External register TRCSTATR bits [31:0] are architecturally mapped to AArch64 System register [TRCSTATR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCSTATR are RES0.

Attributes

TRCSTATR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---|---|---|---|---|---|----------|------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | PMSTABLE | IDLE |

Bits [31:2]

Reserved, RES0.

PMSTABLE, bit [1]

Programmers' model stable.

| PMSTABLE | Meaning |
|----------|---------------------------------------|
| 0b0 | The programmers' model is not stable. |
| 0b1 | The programmers' model is stable. |

Accessing this field has the following behavior:

- When the trace unit is enabled, access to this field is **UNKNOWN/WI**.
- Otherwise, access to this field is **RO**.

IDLE, bit [0]

Idle status.

| IDLE | Meaning |
|------|-----------------------------|
| 0b0 | The trace unit is not idle. |
| 0b1 | The trace unit is idle. |

Accessing TRCSTATR

TRCSTATR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x00C | TRCSTATR |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RO**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCSYNCPR, Synchronization Period Register

The TRCSYNCPR characteristics are:

Purpose

Controls how often trace protocol synchronization requests occur.

Configuration

External register TRCSYNCPR bits [31:0] are architecturally mapped to AArch64 System register [TRCSYNCPR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCSYNCPR are RES0.

Attributes

TRCSYNCPR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|--------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | PERIOD | | | | | | | | | | | | | |

Bits [31:5]

Reserved, RES0.

PERIOD, bits [4:0]

Defines the number of bytes of trace between each periodic trace protocol synchronization request.

| PERIOD | Meaning |
|---------|--|
| 0b00000 | Trace protocol synchronization is disabled. |
| 0b01000 | Trace protocol synchronization request occurs after 2^8 bytes of trace. |
| 0b01001 | Trace protocol synchronization request occurs after 2^9 bytes of trace. |
| 0b01010 | Trace protocol synchronization request occurs after 2^{10} bytes of trace. |
| 0b01011 | Trace protocol synchronization request occurs after 2^{11} bytes of trace. |
| 0b01100 | Trace protocol synchronization request occurs after 2^{12} bytes of trace. |
| 0b01101 | Trace protocol synchronization request occurs after 2^{13} bytes of trace. |
| 0b01110 | Trace protocol synchronization request occurs after 2^{14} bytes of trace. |
| 0b01111 | Trace protocol synchronization request occurs after 2^{15} bytes of trace. |
| 0b10000 | Trace protocol synchronization request occurs after 2^{16} bytes of trace. |
| 0b10001 | Trace protocol synchronization request occurs after 2^{17} bytes of trace. |
| 0b10010 | Trace protocol synchronization request occurs after 2^{18} bytes of trace. |
| 0b10011 | Trace protocol synchronization request occurs after 2^{19} bytes of trace. |
| 0b10100 | Trace protocol synchronization request occurs after 2^{20} bytes of trace. |

Other values are reserved. If a reserved value is programmed into PERIOD, then the behavior of the synchronization period counter is CONSTRAINED UNPREDICTABLE and one of the following behaviors occurs:

- No trace protocol synchronization requests are generated by this counter.
- Trace protocol synchronization requests occur at the specified period.
- Trace protocol synchronization requests occur at some other UNKNOWN period which can vary.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCSYNCP

Must be programmed if [TRCIDR3](#).SYNCP == 0.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCSYNCP can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|----------|
| ETE | 0x034 | TRCSYNCP |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

TRCTRACEIDR, Trace ID Register

The TRCTRACEIDR characteristics are:

Purpose

Sets the trace ID for instruction trace.

Configuration

External register TRCTRACEIDR bits [31:0] are architecturally mapped to AArch64 System register [TRCTRACEIDR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCTRACEIDR are RES0.

Attributes

TRCTRACEIDR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|---|---|---------|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | | | | | | | | | | | | | TRACEID | | | | | | | |

Bits [31:7]

Reserved, RES0.

TRACEID, bits [6:0]

Trace ID field. Sets the trace ID value for instruction trace. The width of the field is indicated by the value of [TRCIDR5](#).TRACEIDSIZE. Unimplemented bits are RES0.

If an implementation supports AMBA ATB, then:

- The width of the field is 7 bits.
- Writing a reserved trace ID value does not affect behavior of the trace unit but it might cause UNPREDICTABLE behavior of the trace capture infrastructure.

See the AMBA ATB Protocol Specification for information about which ATID values are reserved.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCTRACEIDR

Must be programmed if implemented.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCTRACEIDR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-------------|
| ETE | 0x040 | TRCTRACEIDR |

This interface is accessible as follows:

- When OSLockStatus(), or !IsTraceCorePowered() or !AllowExternalTraceAccess() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCTSCTLR, Timestamp Control Register

The TRCTSCTLR characteristics are:

Purpose

Controls the insertion of global timestamps in the trace stream.

Configuration

External register TRCTSCTLR bits [31:0] are architecturally mapped to AArch64 System register [TRCTSCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR0.TSSIZE != 0b00000. Otherwise, direct accesses to TRCTSCTLR are RES0.

Attributes

TRCTSCTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|----|----|----|----|------------|----|----|----|------|----|-----------|----|----|----|---|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | | | | | EVENT_TYPE | | | | RES0 | | EVENT_SEL | | | | | | | | | | | | | |

Bits [31:8]

Reserved, RES0.

EVENT_TYPE, bit [7]
When TRCIDR4.NUMRSPAIR != 0b0000:

Chooses the type of Resource Selector.

| EVENT_TYPE | Meaning |
|------------|--|
| 0b0 | A single Resource Selector. TRCTSCTLR.EVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCTSCTLR.EVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCTSCTLR.EVENT.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EVENT_SEL, bits [4:0]**When TRCIDR4.NUMRSPAIR != 0b0000:**

Defines the selected Resource Selector or pair of Resource Selectors. TRCTSCTLR.EVENT.TYPE controls whether TRCTSCTLR.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCTSCTLR

Must be programmed if [TRCCONFIGR.TS](#) == 1.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCTSCTLR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| ETE | 0x030 | TRCTSCTLR |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVICTLR, ViewInst Main Control Register

The TRCVICTLR characteristics are:

Purpose

Controls instruction trace filtering.

Configuration

External register TRCVICTLR bits [31:0] are architecturally mapped to AArch64 System register [TRCVICTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented. Otherwise, direct accesses to TRCVICTLR are RES0.

Attributes

TRCVICTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | |
|------|----------------|----------------|----------------|------|----------------|----------------|----------------|----------------|----------------|----------------|------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 |
| RES0 | EXLEVEL_RL_EL2 | EXLEVEL_RL_EL1 | EXLEVEL_RL_EL0 | RES0 | EXLEVEL_NS_EL2 | EXLEVEL_NS_EL1 | EXLEVEL_NS_EL0 | EXLEVEL_RL_EL2 | EXLEVEL_RL_EL1 | EXLEVEL_RL_EL0 | RES0 |

Bits [31:27]

Reserved, RES0.

EXLEVEL_RL_EL2, bit [26]

When TRCIDR6.EXLEVEL_RL_EL2 == 1:

Filter instruction trace for EL2 in Realm state.

| EXLEVEL_RL_EL2 | Meaning |
|----------------|--|
| 0b0 | When TRCVICTLR.EXLEVEL_NS_EL2 is 0 the trace unit generates instruction trace for EL2 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL2 is 1 the trace unit does not generate instruction trace for EL2 in Realm state. |
| 0b1 | When TRCVICTLR.EXLEVEL_NS_EL2 is 0 the trace unit does not generate instruction trace for EL2 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL2 is 1 the trace unit generates instruction trace for EL2 in Realm state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL1, bit [25]**When TRCIDR6.EXLEVEL_RL_EL1 == 1:**

Filter instruction trace for EL1 in Realm state.

| EXLEVEL_RL_EL1 | Meaning |
|----------------|--|
| 0b0 | When TRCVICTLR.EXLEVEL_NS_EL1 is 0 the trace unit generates instruction trace for EL1 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL1 is 1 the trace unit does not generate instruction trace for EL1 in Realm state. |
| 0b1 | When TRCVICTLR.EXLEVEL_NS_EL1 is 0 the trace unit does not generate instruction trace for EL1 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL1 is 1 the trace unit generates instruction trace for EL1 in Realm state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_RL_EL0, bit [24]**When TRCIDR6.EXLEVEL_RL_EL0 == 1:**

Filter instruction trace for EL0 in Realm state.

| EXLEVEL_RL_EL0 | Meaning |
|----------------|--|
| 0b0 | When TRCVICTLR.EXLEVEL_NS_EL0 is 0 the trace unit generates instruction trace for EL0 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL0 is 1 the trace unit does not generate instruction trace for EL0 in Realm state. |
| 0b1 | When TRCVICTLR.EXLEVEL_NS_EL0 is 0 the trace unit does not generate instruction trace for EL0 in Realm state. When TRCVICTLR.EXLEVEL_NS_EL0 is 1 the trace unit generates instruction trace for EL0 in Realm state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bit [23]

Reserved, RES0.

EXLEVEL_NS_EL2, bit [22]**When Non-secure EL2 is implemented:**

Filter instruction trace for EL2 in Non-secure state.

| EXLEVEL_NS_EL2 | Meaning |
|----------------|---|
| 0b0 | The trace unit generates instruction trace for EL2 in Non-secure state. |
| 0b1 | The trace unit does not generate instruction trace for EL2 in Non-secure state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_EL1, bit [21]

When Non-secure EL1 is implemented:

Filter instruction trace for EL1 in Non-secure state.

| EXLEVEL_NS_EL1 | Meaning |
|----------------|---|
| 0b0 | The trace unit generates instruction trace for EL1 in Non-secure state. |
| 0b1 | The trace unit does not generate instruction trace for EL1 in Non-secure state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_NS_ELO, bit [20]

When Non-secure ELO is implemented:

Filter instruction trace for ELO in Non-secure state.

| EXLEVEL_NS_ELO | Meaning |
|----------------|---|
| 0b0 | The trace unit generates instruction trace for ELO in Non-secure state. |
| 0b1 | The trace unit does not generate instruction trace for ELO in Non-secure state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL3, bit [19]

When EL3 is implemented:

Filter instruction trace for EL3.

| EXLEVEL_S_EL3 | Meaning |
|---------------|---|
| 0b0 | The trace unit generates instruction trace for EL3. |
| 0b1 | The trace unit does not generate instruction trace for EL3. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL2, bit [18]

When EL2 is implemented and FEAT_SEL2 is implemented:

Filter instruction trace for EL2 in Secure state.

| EXLEVEL_S_EL2 | Meaning |
|---------------|---|
| 0b0 | The trace unit generates instruction trace for EL2 in Secure state. |
| 0b1 | The trace unit does not generate instruction trace for EL2 in Secure state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL1, bit [17]

When Secure EL1 is implemented:

Filter instruction trace for EL1 in Secure state.

| EXLEVEL_S_EL1 | Meaning |
|---------------|---|
| 0b0 | The trace unit generates instruction trace for EL1 in Secure state. |
| 0b1 | The trace unit does not generate instruction trace for EL1 in Secure state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

EXLEVEL_S_EL0, bit [16]

When Secure EL0 is implemented:

Filter instruction trace for EL0 in Secure state.

| EXLEVEL S_ELO | Meaning |
|---------------|---|
| 0b0 | The trace unit generates instruction trace for ELO in Secure state. |
| 0b1 | The trace unit does not generate instruction trace for ELO in Secure state. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [15:12]

Reserved, RES0.

TRCERR, bit [11]

When TRCIDR3.TRCERR == 1:

Controls the forced tracing of System Error exceptions.

| TRCERR | Meaning |
|--------|--|
| 0b0 | Forced tracing of System Error exceptions is disabled. |
| 0b1 | Forced tracing of System Error exceptions is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

TRCRESET, bit [10]

Controls the forced tracing of PE Resets.

| TRCRESET | Meaning |
|----------|--|
| 0b0 | Forced tracing of PE Resets is disabled. |
| 0b1 | Forced tracing of PE Resets is enabled. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

SSSTATUS, bit [9]

ViewInst start/stop function status.

| SSSTATUS | Meaning |
|----------|---|
| 0b0 | Stopped State. The ViewInst start/stop function is in the stopped state. |
| 0b1 | Started State. The ViewInst start/stop function is in the started state. |

Before software enables the trace unit, it must write to this field to set the initial state of the ViewInst start/stop function. If the ViewInst start/stop function is not used then set this field to 1. Arm recommends that the value of this field is set before each trace session begins.

If the trace unit becomes disabled while a start point or stop point is still speculative, then the value of TRCVICTLR.SSSTATUS is UNKNOWN and might represent the result of a speculative start point or stop point.

If software which is running on the PE being traced disables the trace unit, either by clearing [TRCPRGCTLR.EN](#) or locking the OS Lock, Arm recommends that a DSB and an ISB instruction are executed before disabling the trace unit to prevent any start points or stop points being speculative at the point of disabling the trace unit. This procedure assumes that all start points or stop points occur before the barrier instructions are executed. The procedure does not guarantee that there are no speculative start points or stop points when disabling, although it helps minimize the probability.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing this field has the following behavior:

- RES1** if all of the following are true:
 - TRCIDR4.NUMACPAIRS == 0b0000
 - TRCIDR4.NUMPC == 0b0000
- Otherwise, access to this field is **RW**.

Bit [8]

Reserved, RES0.

EVENT_TYPE, bit [7]

When TRCIDR4.NUMRSPAIR != 0b0000:

Chooses the type of Resource Selector.

| EVENT_TYPE | Meaning |
|------------|--|
| 0b0 | A single Resource Selector. TRCVICTLR.EVENT.SEL[4:0] selects the single Resource Selector, from 0-31, used to activate the resource event. |
| 0b1 | A Boolean-combined pair of Resource Selectors. TRCVICTLR.EVENT.SEL[3:0] selects the Resource Selector pair, from 0-15, that has a Boolean function that is applied to it whose output is used to activate the resource event. TRCVICTLR.EVENT.SEL[4] is RES0. |

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Bits [6:5]

Reserved, RES0.

EVENT_SEL, bits [4:0]

When TRCIDR4.NUMRSPAIR != 0b0000:

Defines the selected Resource Selector or pair of Resource Selectors. TRCVICTLR.EVENT.TYPE controls whether TRCVICTLR.EVENT.SEL is the index of a single Resource Selector, or the index of a pair of Resource Selectors.

If an unimplemented Resource Selector is selected using this field, the behavior of the resource event is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

Selecting Resource Selector pair 0 using this field is UNPREDICTABLE, and the resource event might fire or might not fire when the resources are not in the Paused state.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

When TRCIDR4.NUMRSPAIR == 0b0000:

This field is reserved:

- Bits [4:1] are RES0.
- Bit [0] is RES1.

Otherwise:

Reserved, RES0.

Accessing TRCVICTLR

Must be programmed.

Reads from this register might return an UNKNOWN value if the trace unit is not in either of the Idle or Stable states.

TRCVICTLR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-----------|
| ETE | 0x080 | TRCVICTLR |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVIIECTLR, ViewInst Include/Exclude Control Register

The TRCVIIECTLR characteristics are:

Purpose

Use this to select, or read, the Address Range Comparators for the ViewInst include/exclude function.

Configuration

External register TRCVIIECTLR bits [31:0] are architecturally mapped to AArch64 System register [TRCVIIECTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMACPAIRS > 0b0000. Otherwise, direct accesses to TRCVIIECTLR are RES0.

Attributes

TRCVIIECTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|------------|----|------------|----|------------|----|------------|----|------------|----|------------|----|------------|----|------------|---|---|---|---|---|---|---|---|---|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| RES0 | | | | | | | | EXCLUDE[7] | | EXCLUDE[6] | | EXCLUDE[5] | | EXCLUDE[4] | | EXCLUDE[3] | | EXCLUDE[2] | | EXCLUDE[1] | | EXCLUDE[0] | | | | | | | | | |

Bits [31:24]

Reserved, RES0.

EXCLUDE[<m>], bit [m+16], for m = 7 to 0

Exclude Address Range Comparator <m>. Selects whether Address Range Comparator <m> is in use with the ViewInst exclude function.

| EXCLUDE[<m>] | Meaning |
|--------------|---|
| 0b0 | The address range that Address Range Comparator <m> defines, is not selected for the ViewInst exclude function. |
| 0b1 | The address range that Address Range Comparator <m> defines, is selected for the ViewInst exclude function. |

This bit is RES0 if m >= [TRCIDR4](#).NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [15:8]

Reserved, RES0.

INCLUDE[<m>], bit [m], for m = 7 to 0

Include Address Range Comparator <m>.

Selects whether Address Range Comparator <m> is in use with the ViewInst include function.

Selecting no comparators for the ViewInst include function indicates that all instructions are included by default.

The ViewInst exclude function then indicates which ranges are excluded.

| INCLUDE[<m>] | Meaning |
|--------------|---|
| 0b0 | The address range that Address Range Comparator <m> defines, is not selected for the ViewInst include function. |
| 0b1 | The address range that Address Range Comparator <m> defines, is selected for the ViewInst include function. |

This bit is RES0 if m >= [TRCIDR4](#).NUMACPAIRS.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCVIIECTLR

Must be programmed if [TRCIDR4](#).NUMACPAIRS > 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCVIIECTLR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-------------|
| ETE | 0x084 | TRCVIIECTLR |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVIPCSSCTLR, ViewInst Start/Stop PE Comparator Control Register

The TRCVIPCSSCTLR characteristics are:

Purpose

Use this to select, or read, which PE Comparator Inputs can control the ViewInst start/stop function.

Configuration

External register TRCVIPCSSCTLR bits [31:0] are architecturally mapped to AArch64 System register [TRCVIPCSSCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMPC > 0b0000. Otherwise, direct accesses to TRCVIPCSSCTLR are RES0.

Attributes

TRCVIPCSSCTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | |
|------|----|----|----|----|----|----|----|---------|---------|---------|---------|---------|---------|---------|---------|------|----|----|----|----|----|---|---|----------|----------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 |
| RES0 | | | | | | | | STOP[7] | STOP[6] | STOP[5] | STOP[4] | STOP[3] | STOP[2] | STOP[1] | STOP[0] | RES0 | | | | | | | | START[7] | START[6] |

Bits [31:24]

Reserved, RES0.

STOP[<m>], bit [m+16], for m = 7 to 0

Selects whether PE Comparator Input <m> is in use with ViewInst start/stop function, for the purpose of stopping trace.

| STOP[<m>] | Meaning |
|-----------|--|
| 0b0 | The PE Comparator Input <m>, is not selected as a stop resource. |
| 0b1 | The PE Comparator Input <m>, is selected as a stop resource. |

This bit is RES0 if m >= [TRCIDR4](#).NUMPC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Bits [15:8]

Reserved, RES0.

START[<m>], bit [m], for m = 7 to 0

Selects whether PE Comparator Input <m> is in use with ViewInst start/stop function, for the purpose of starting trace.

| START[<m>] | Meaning |
|------------|---|
| 0b0 | The PE Comparator Input <m>, is not selected as a start resource. |
| 0b1 | The PE Comparator Input <m>, is selected as a start resource. |

This bit is RES0 if m >= [TRCIDR4](#).NUMPC.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCVIPCSSCTLR

Must be programmed if [TRCIDR4](#).NUMPC != 0b0000.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCVIPCSSCTLR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|---------------|
| ETE | 0x08C | TRCVIPCSSCTLR |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVISSCTLR, ViewInst Start/Stop Control Register

The TRCVISSCTLR characteristics are:

Purpose

Use this to select, or read, the Single Address Comparators for the ViewInst start/stop function.

Configuration

External register TRCVISSCTLR bits [31:0] are architecturally mapped to AArch64 System register [TRCVISSCTLR\[31:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMACPAIRS > 0b0000. Otherwise, direct accesses to TRCVISSCTLR are RES0.

Attributes

TRCVISSCTLR is a 32-bit register.

Field descriptions

| | | | | | | | | | | | | |
|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|--------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|-------------------------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 |
| STOP[15] | STOP[14] | STOP[13] | STOP[12] | STOP[11] | STOP[10] | STOP[9] | STOP[8] | STOP[7] | STOP[6] | STOP[5] | STOP[4] | STOP[3] |

STOP[<m>], bit [m+16], for m = 15 to 0

Selects whether Single Address Comparator <m> is used with the ViewInst start/stop function, for the purpose of stopping trace.

| STOP[<m>] | Meaning |
|-----------|--|
| 0b0 | The Single Address Comparator <m>, is not selected as a stop resource. |
| 0b1 | The Single Address Comparator <m>, is selected as a stop resource. |

This bit is RES0 if $m \geq 2 \times \text{TRCIDR4.NUMACPAIRS}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

START[<m>], bit [m], for m = 15 to 0

Selects whether Single Address Comparator <m> is used with the ViewInst start/stop function, for the purpose of starting trace.

| START[<m>] | Meaning |
|------------|---|
| 0b0 | The Single Address Comparator <m>, is not selected as a start resource. |
| 0b1 | The Single Address Comparator <m>, is selected as a start resource. |

This bit is RES0 if $m \geq 2 \times \text{TRCIDR4.NUMACPAIRS}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCVISSCTLR

Must be programmed if [TRCIDR4](#).NUMACPAIRS > 0b0000.

For any 2 comparators selected for the ViewInst start/stop function, the comparator containing the lower address must be a lower numbered comparator.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCVISSCTLR can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|-------------|
| ETE | 0x088 | TRCVISSCTLR |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVMIDCCTRL0, Virtual Context Identifier Comparator Control Register 0

The TRCVMIDCTRL0 characteristics are:

Purpose

Virtual Context Identifier Comparator mask values for the [TRCVMIDCVR<n>](#) registers, where n=0-3.

Configuration

External register TRCVMIDCCTLRO bits [31:0] are architecturally mapped to AArch64 System register [TRCVMIDCCTLRO\[31:0\]](#).

This register is present only when FEAT_ETE is implemented, TRCIDR4.NUMVMIDC > 0x0 and TRCIDR2.VMIDSIZE > 0b000000. Otherwise, direct accesses to TRCVMIDCCTLR0 are RES0.

Attributes

TRCVMIDCTL0 is a 32-bit register.

Field descriptions

| | | | | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 |
| COMP3[7] | COMP3[6] | COMP3[5] | COMP3[4] | COMP3[3] | COMP3[2] | COMP3[1] | COMP3[0] | COMP2[7] | COMP2[6] | COMP2[5] |

COMP3[<m>], bit [m+24], for m = 7 to 0

When TRCIDR4.NUMVMIDC > 3:

TRCVMIDCVR3 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR3. Each bit in this field corresponds to a byte in TRCVMIDCVR3.

| COMP3[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR3[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if $m \geq \text{TRCIDR2.VMIDSIZE}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP2[<m>], bit [m+16], for m = 7 to 0

When TRCIDR4.NUMVMIDC > 2:

TRCVMIDCVR2 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR2. Each bit in this field corresponds to a byte in TRCVMIDCVR2.

| COMP2[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR2[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR2[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.VMIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP1[<m>], bit [m+8], for m = 7 to 0

When TRCIDR4.NUMVMIDC > 1:

TRCVMIDCVR1 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR1. Each bit in this field corresponds to a byte in TRCVMIDCVR1.

| COMP1[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR1[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR1[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.VMIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP0[<m>], bit [m], for m = 7 to 0

When TRCIDR4.NUMVMIDC > 0:

TRCVMIDCVR0 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR0. Each bit in this field corresponds to a byte in TRCVMIDCVR0.

| COMP0[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR0[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR0[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.VMIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCVMIDCCTLRO

If software uses the [TRCVMIDCVR<n>](#) registers, where n=0-3, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in [TRCVMIDCVR<n>](#) to 0x00.

If any bit is 1 and the relevant byte in [TRCVMIDCVR<n>](#) is not 0x00, the behavior of the Virtual Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCVMIDCCTLRO can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|---------------|
| ETE | 0x688 | TRCVMIDCCTLRO |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVMIDCTRL1, Virtual Context Identifier Comparator Control Register 1

The TRCVMIDCCTLR1 characteristics are:

Purpose

Virtual Context Identifier Comparator mask values for the [TRCVMIDCVR<n>](#) registers, where n=4-7.

Configuration

External register TRCVMIDCCTL1 bits [31:0] are architecturally mapped to AArch64 System register [TRCVMIDCCTL1\[31:0\]](#).

This register is present only when FEAT_ETE is implemented, TRCIDR4.NUMVMIDC > 0x4 and TRCIDR2.VMIDSIZE > 0b00000. Otherwise, direct accesses to TRCVMIDCCTLR1 are RES0.

Attributes

TRCVMIDCCTL1 is a 32-bit register.

Field descriptions

| | | | | | | | | | | |
|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|----------|
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 |
| COMP7[7] | COMP7[6] | COMP7[5] | COMP7[4] | COMP7[3] | COMP7[2] | COMP7[1] | COMP7[0] | COMP6[7] | COMP6[6] | COMP6[5] |

COMP7[<m>], bit [m+24], for m = 7 to 0
When TRCIDR4.NUMVMIDC > 7:

TRCVMIDCVR7 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR7. Each bit in this field corresponds to a byte in TRCVMIDCVR7.

| COMP7[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR7[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if $m \geq \text{TRCIDR2.VMIDSIZE}$.

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP6[<m>], bit [m+16], for m = 7 to 0
When TRCIDR4.NUMVMIDC > 6:

TRCVMIDCVR6 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR6. Each bit in this field corresponds to a byte in TRCVMIDCVR6.

| COMP6[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR6[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR6[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.VMIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP5[<m>], bit [m+8], for m = 7 to 0

When TRCIDR4.NUMVMIDC > 5:

TRCVMIDCVR5 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR5. Each bit in this field corresponds to a byte in TRCVMIDCVR5.

| COMP5[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR5[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR5[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.VMIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

COMP4[<m>], bit [m], for m = 7 to 0

When TRCIDR4.NUMVMIDC > 4:

TRCVMIDCVR4 mask control. Specifies the mask value that the trace unit applies to TRCVMIDCVR4. Each bit in this field corresponds to a byte in TRCVMIDCVR4.

| COMP4[<m>] | Meaning |
|------------|--|
| 0b0 | The trace unit includes TRCVMIDCVR4[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |
| 0b1 | The trace unit ignores TRCVMIDCVR4[(m×8+7):(m×8)] when it performs the Virtual context identifier comparison. |

This bit is RES0 if m >= [TRCIDR2.VMIDSIZE](#).

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Otherwise:

Reserved, RES0.

Accessing TRCVMIDCCTLR1

If software uses the [TRCVMIDCVR<n>](#) registers, where n=4-7, then it must program this register.

If software sets a mask bit to 1 then it must program the relevant byte in [TRCVMIDCVR<n>](#) to 0x00.

If any bit is 1 and the relevant byte in [TRCVMIDCVR<n>](#) is not 0x00, the behavior of the Virtual Context Identifier Comparator is CONSTRAINED UNPREDICTABLE. In this scenario the comparator might match unexpectedly or might not match.

Writes are CONSTRAINED UNPREDICTABLE if the trace unit is not in the Idle state.

TRCVMIDCCTLR1 can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|--------|---------------|
| ETE | 0x68C | TRCVMIDCCTLR1 |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

30/09/2021 14:53; 092b4e1bbfbb45a293b198f9330c5f529ead2b0f

Copyright © 2010-2021 Arm Limited or its affiliates. All rights reserved. This document is Non-Confidential.

TRCVMIDCVR<n>, Virtual Context Identifier Comparator Value Register <n>, n = 0 - 7

The TRCVMIDCVR<n> characteristics are:

Purpose

Contains the Virtual Context Identifier Comparator value.

Configuration

External register TRCVMIDCVR<n> bits [63:0] are architecturally mapped to AArch64 System register [TRCVMIDCVR<n>\[63:0\]](#).

This register is present only when FEAT_ETE is implemented and TRCIDR4.NUMVMIDC > n. Otherwise, direct accesses to TRCVMIDCVR<n> are RES0.

Attributes

TRCVMIDCVR<n> is a 64-bit register.

Field descriptions

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|-------|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 63 | 62 | 61 | 60 | 59 | 58 | 57 | 56 | 55 | 54 | 53 | 52 | 51 | 50 | 49 | 48 | 47 | 46 | 45 | 44 | 43 | 42 | 41 | 40 | 39 | 38 | 37 | 36 | 35 | 34 | 33 | 32 |
| | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |
| | | | | | | | | | | | | | | | | VALUE | | | | | | | | | | | | | | | |
| 31 | 30 | 29 | 28 | 27 | 26 | 25 | 24 | 23 | 22 | 21 | 20 | 19 | 18 | 17 | 16 | 15 | 14 | 13 | 12 | 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |

VALUE, bits [63:0]

Virtual context identifier value. The width of this field is indicated by [TRCIDR2.VMIDSIZE](#). Unimplemented bits are RES0. After a PE Reset, the trace unit assumes that the Virtual context identifier is zero until the PE updates the Virtual context identifier .

The reset behavior of this field is:

- On a Trace unit reset, this field resets to an architecturally UNKNOWN value.

Accessing TRCVMIDCVR<n>

Must be programmed if any of the following are true:

- [TRCRSCTLR<a>](#).GROUP == 0b0111 and [TRCRSCTLR<a>](#).VMID[n] == 1.
- [TRCACATR<a>](#).CONTEXTTYPE == 0b10 or 0b11 and [TRCACATR<a>](#).CONTEXT == n.

TRCVMIDCVR<n> can be accessed through the external debug interface:

| Component | Offset | Instance |
|-----------|-----------------|---------------|
| ETE | 0x640 + (8 * n) | TRCVMIDCVR<n> |

This interface is accessible as follows:

- When OSLockStatus(), or !AllowExternalTraceAccess() or !IsTraceCorePowered() accesses to this register generate an error response.
- Otherwise accesses to this register are **RW**.

